(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 811 931 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
10.12.1997 Bulletin 1997/50

(51) Int Cl.6: G06F 13/40

(21) Application number: 97303797.1

(22) Date of filing: 04.06.1997

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

(30) Priority: 05.06.1996 US 658602

(71) Applicant: Compaq Computer Corporation
Houston Texas 77070 (US)

(72) Inventors:
- Culley, Paul R.
Cypress,Texas 77429 (US)
- Goodrum, Alan L.
Tomball, Texas 77375 (US)
- Chow, Raymond Y. L.
Cypress, Texas 77429 (US)
- Basile, Barry S.
Houston, Texas 77084 (US)

(74) Representative: Brunner, Michael John
GILL JENNINGS & EVERY
Broadgate House
7 Eldon Street
London EC2M 7LH (GB)

(54) Expansion card insertion and removal

(57) A computer system has a bus, a connector for a circuit card, and a clamp configured to selectively prevent removal of the circuit card from the connector when the clamp is engaged. The computer system has circuitry connected to monitor the engagement status of the clamp and to regulate delivery of power to the connector based on the engagement state of the clamp.

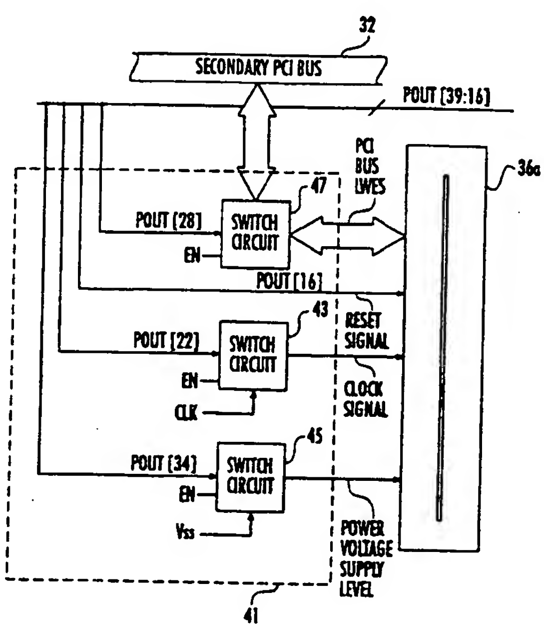


FIG. 28

EP 0 811 931 A2

essary powering down of the expansion card. The control circuit has control of the expansion bus during critical phases of the power up and power down sequences, which promotes bus integrity.

Other advantages and features will become apparent from the following description and from the claims.

Description

Figure 1 is a block diagram of a computer system.

Figure 2 is a block diagram of an expansion box of the computer system of Figure 1.

Figure 3 is a block diagram of the bridge chips in the computer system.

Figure 4 is a block diagram of a queue block in each of the bridge chips.

Figure 5 is a block diagram of the clock routing scheme in the bridge chips.

Figure 6 is a block diagram of a clock generator in each of the bridge chips.

Figure 7 is a block diagram of a master cable interface in each of the bridge chips for transmitting data over a cable connecting the bridge chips.

Figure 8 is a timing diagram of signals in the master cable interface.

Figure 9 is a block diagram of a slave cable interface in each of the bridge chips for receiving data transmitted over the cable.

Figure 10 is a block diagram of logic generating input and output pointers for the receiving logic in the slave cable interface.

Figure 11 is a timing diagram of signals in the slave cable interface.

Figure 12 is a timing diagram of the input and output pointers and their relation to the received cable data.

Figure 13 is a block diagram of the placement of flip flops and input and output pads in each of the bridge chips.

Figure 14 is a table of the information carried by the cable.

Figure 15A is a table showing the type of information carried by the cable signals associated with single address cycle transactions.

Figure 15B is a table showing the type of information carried by the cable signals associated with dual-address cycle transactions.

Figure 16 is a table of parameters associated with the cable.

Figure 17 is a logic diagram of an error detection and correction circuit.

Figure 18 is a parity-check matrix for generating check bits in the error detection and correction circuit.

Figure 19 is a syndrome table for generating fix bits in the error detection and correction circuit.

Figure 20A is a state diagram showing a round-robin arbitration scheme.

Figure 20B is a state diagram showing a two-level arbitration scheme.

Figure 21 is a logic diagram of an arbiter in each of the bridge chips.

Figure 22 is a state diagram of a grant state machine in an arbiter.

Figure 23 is a state diagram of a level one arbitration state machine in the arbiter.

Figure 24 is a table showing generation of new grant signals based on the current master.

Figure 25 is a block diagram of logic for generating mask bits and multi-threaded master indication bits.

Figure 26A is a logic diagram of circuits for generating the masked bits.

Figure 26B is a block diagram of a computer system with multiple layers of buses.

Figure 27A is a side view of an expansion card inserted into a slot.

Figure 27B-C are schematic diagrams of lever circuitry.

Figures 28-31 are schematic diagrams of circuitry of the expansion box.

Figure 32A is a state diagram from the circuitry of the expansion box.

Figure 32B are waveforms from the circuitry of the expansion box.

Figure 33A is a schematic diagram of circuitry of the expansion box.

Figure 33B are waveforms from the circuitry of the expansion box.

Figures 33C-H are a state diagram from the circuitry of the expansion box.

Figure 34 is a schematic diagram of circuitry of the expansion box.

Figure 35A is a state diagram from the circuitry of the expansion box.

Figure 35B are waveforms from the circuitry of the expansion box.

Figure 36 is a schematic diagram of circuitry of the expansion box.

Figure 37 is a flow diagram of a non-maskable interrupt handler invoked in response to detection of a bus hang condition in the computer system.

Figure 38 is a flow diagram of a BIOS routine that is invoked by a computer system lock-up event.

Figure 39 is a flow diagram of a BIOS isolate routine invoked in response to a bus-hang condition or the computer lock-up event.

Figure 40 is a block diagram of a bus watcher in each of the bridge chips.

ventional expansion cards 807 (Fig. 27A). One slot 34 on the expansion box receives a card 46 which has the bridge chip 26. Each hot-plug slot 36 has associated switch circuitry 41 for connecting and disconnecting the slot 36 to and from the PCI bus 32. Six mechanical levers 802 are used to selectively secure (when closed or latched) the cards 807 to corresponding slots, as further described in U.S. Patent Application Serial No. 08/658385, entitled "Securing a Card in an Electronic Device," filed on the same date as this application and incorporated by reference. Each expansion box 30 includes registers 52 and 82 for monitoring the levers 802 and status signals of the expansion box 30 and a register 80 for controlling connection and disconnection of slots 36 to the PCI bus 32.

Referring to Figure 3, the bridge chip is designed to be used in pairs 26 and 48 to form a PCI-PCI bridge between the primary PCI bus 24 and the secondary PCI bus 32. The programming model is that of two hierarchical bridges. To the system software, the cable 28 appears as a PCI bus which contains exactly one device, the downstream bridge chip 48. This greatly simplifies the configuration of the 2-chip PCI-PCI bridge joining the primary and secondary buses. The bridge chip 26, which is closer to the CPU 14, joins the primary PCI bus 24 to the cable 28. The second PCI-PCI bridge 48 resides in the expansion box 30 and joins the cable 28 to the secondary PCI bus 32. A mode pin UPSTREAM_CHIP determines whether the bridge chip operates in the upstream mode or the downstream mode. Some non-bridge functions such as a bus monitor 106 and hot plug logic in an SIO 50 are used only in the expansion box 30, and are non-functional in the upstream mode chip 26.

A clock generator 102 in the bridge chip 26 generates clocks based on the clock PCICLK1 on the primary PCI bus 24, with one of the generated clocks being provided through the cable 28 to a clock generator 122 in the downstream bridge chip 48. The clock generator 122 generates and drives the PCI clocks in the expansion box 30 at the same frequency of the primary PCI bus 24, which results in both bridge chips 26 and 48 being run at the same frequency. The downstream bridge chip 48 lags the upstream bridge chip 26 in phase by the delay of the cable 28. An asynchronous boundary in the upstream bridge chip 26 at the point where data is taken off of the cable 28 allows the phase delay to be any value (and therefore the cable to be of any length), with the only requirement only being that the frequency of the two bridge chips be the same.

The core logic of each bridge chip is the bridge logic block (100 or 120), which includes a PCI master (101 or 123) for acting as a master on the respective PCI bus, a PCI target or slave (103 or 121) for acting as a slave device on the respective PCI bus, configuration registers (105 or 125) which contain the configuration information of the corresponding bridge chip, and a queue block (107 or 127) containing several queues in which data associated with transactions between the primary PCI bus and the secondary PCI bus 32 are queued and managed. The data transferred between the upstream bridge chip 26 and the downstream bridge chip 48 are buffered by cable interfaces 104 and 130 in the bridge chips 26 and 48, respectively.

Interrupt routing logic is also included in each bridge chip. There are 8 interrupts, 6 from the secondary bus slots, 1 from an SIO circuit 50, and 1 from the downstream bridge chip 48. In the downstream chip 48, the interrupts are received by an interrupt receiving block 132 and sent up the cable 28 as a serial stream in sequential time slices. In the upstream bridge chip 26, the interrupts are received by an interrupt output block 114, which routes the interrupts to an interrupt controller.

The SIO circuit 50 furnishes control signals for lighting LEDs, for controlling reset, and for selectively connecting the slots 36 to the bus 32. It also includes logic for reading the engagement status of the levers 802, and the status of the cards 807 in each slot 36.

The bridge circuit 26 also includes support for interrupts in the expansion box 30, and, when installed in a slot in the host system with a proprietary interface to a multichannel interrupt controller, it sends the states of each interrupt in a serial stream. The bridge circuit 26 also can be configured to drive standard PCI INTA, INTB, INTC, and INTD signals if it is installed in a standard slot in the host system.

Each bridge chip also includes a PCI arbiter (116 or 124) for controlling access to up to seven bus masters. As the upstream bridge 26 is installed in a slot, the PCI arbiter 116 in the upstream bridge chip 26 is disabled. Each bridge chip also includes an I²C controller (108 or 126) for communication with devices such as EEPROMs, temperature sensors, and so forth, a JTAG master (110 or 128) for performing test cycles, a bus monitor (106 or 127) for measuring bus utilization and efficiency and the efficiency of the bridge chip's prefetch algorithm, and a bus watcher (119 or 129) for storing bus history and state vector information and for notifying the CPU 14 of a bus hang condition. Certain blocks are disabled in each bridge chip as they are not used. In the upstream bridge chip 26, the bus watcher 119, the SIO 118, the PCI arbiter 116, and the bus monitor 106 are disabled. In addition, the interrupt receiving block 112 in the upstream chip 26 and the interrupt output block 134 in the downstream chip 48 are disabled.

QUEUE BLOCK OVERVIEW

Referring to Figure 4, the queue blocks 107 and 127 manage transactions flowing between the primary PCI bus 24 (in the upstream chip) or the secondary PCI bus 32 (in the downstream chip) and the cable interface 130. (From here on, the downstream bridge chip will be referred to with the assumption that upstream chip works identically, unless

contains completion data (i.e., specifically requested by a device that has not yet returned to retrieve it). If the buffer contains completion data and the requesting device has issued a request that does not "hit" the buffer, the DCQ 144 tags the device as a "multi-threaded" device (i.e., one that is capable of maintaining more than one transaction at once) and allocates another completion buffer for the new request. The buffer flushing and multiple buffer allocation schemes are described in more detail below.

A master cycle arbiter (MCA) 150 in the queue block 127 maintains standard ordering constraints between posted memory write, delayed request, and delayed completion transactions, as set forth in the PCI Bridge Architecture Specification, Version 2.1. These constraints require that bus cycles maintain strong write ordering and that deadlocks do not occur. Therefore, the MCA 150 determines the order in which posted memory write transactions in the PMWQ 140 and delayed request transactions in the DRQ 142 are run on the PCI bus 32. The MCA 150 also controls the availability of delayed completion information stored in the DCQ 144. To ensure compliance with these rules, the downstream MCA 150 gives each posted memory write cycle an opportunity to bypass earlier-issued delayed request cycles, while both the downstream and the upstream MCAs 150 do not allow delayed request and delayed completion cycles to bypass earlier-issued posted memory write cycles. Transaction ordering by the MCA 150 is described in more detail below.

The transaction counters 159 in the downstream queue block 127 maintain a count of the number of transactions enqueued in the upstream bridge chip. A posted memory write (PMW) counter 160 indicates the number of PMW transactions held in the upstream posted memory write queue. The PMW counter 160 is incremented each time a PMW transaction is sent to the cable interface 130. The counter 160 is decremented each time the QPIF 148 receives a signal from the cable decoder 146 indicating that a PMW cycle has been completed on the upstream PCI bus 24. When the upstream PMWQ has enqueued the maximum number (four) of PMW transactions, the PMW counter 160 asserts a PMW full signal (tc_pmw_full) that tells the QPIF 148 to retry additional PMW cycles from the PCI bus 32. Likewise, a delayed request (DR) counter 161 counts the number of DR transactions held in the upstream delayed request queue. When the DRQ is holding the maximum number (three) of DR transactions, the DR counter 161 asserts a DR full signal (tc_dr_full) indicating that the QPIF 148 must retry all subsequent DR transactions from the PCI bus 32. A delayed completion (DC) counter 162 counts the number of delayed completions that are enqueued in the upstream master cycle arbiter. When the MCA is holding the maximum number (four) of delayed completions, the DC counter 162 asserts a DC full signal (tc_dc_full) that prevents the downstream QPIF 148 from running delayed request transactions on the secondary PCI bus 32. As soon as the full condition disappears, delayed completion information may be sent to downstream DCQ.

A PCI interface block 152 resides between the PCI bus 32 and the queue block 127. The PCI interface 152 includes a master block 123 and a slave (target) block 121. The slave block 121 allows PCI devices on the bus 32 to access the bridge chip's internal registers (e.g., target memory range registers 155 and configuration registers), to claim completion information stored in the DCQ 144, and to initiate transactions that are passed through the QPIF 148 and the cable interface 130 to the primary bus. The slave block 121 controls the availability of the PCI bus 32 to the PCI devices on the bus 32 by recognizing when each device asserts its REQ# line and forwarding the REQ# signals to the PCI arbiter 124. When the PCI arbiter 124 selects a requesting device to receive control of the bus, the slave block 121 grants the bus to the device by asserting the device's GNT# line. As soon as the bus 32 is granted to the requesting device and the device asserts its FRAME# signal indicating the beginning of a transaction, the slave block 121 latches the transaction information (e.g., address, command, data, byte enables, parity, etc.) into a slave latching register 156. The queue block 127 then is able to retrieve the transaction information from the latching register 156 and provide it to the DCQ 144 and/or the cable interface 130.

Transactions supported by the PCI slave block 121 are shown in the following table.

PCI Interface Slave Transactions		
Transaction Type	Primary Interface	Secondary Interface
Interrupt Acknowledge	Not supported	Not supported
Special Cycle	Delayed	Delayed
I/O Read	Delayed	Delayed
I/O Write	Delayed	Delayed
Memory Read	Delayed	Delayed
Memory Write	Posted	Posted
Configuration Read (type 0)	Immediate	Not supported

that are used by the queue block 127 to flush the completion buffers: a flush signal (p2q_flush) that indicates when a buffer should be flushed, and a slot selection signal (p2q_slot[2:0]) that indicates which PCI device (i.e., which slot on the PCI bus) should have data flushed. The following table shows the relationship between p2q_slot[2:0] and the PCI slot number.

Creation of p2q_slot[2:0]	
p2q_slot[2:0]	slot number
000	all
001	1
010	2
011	3
100	4
101	5
110	6
111	7

When p2q_flush is asserted, the queue block 127 will flush either all of the completion buffers in the DCQ 144 if p2q_slot[2:0] is equal to "000" or the corresponding one of the eight completion buffers if p2q_slot[2:0] has any other value. The queue block 127 keeps track of which completion buffers, if any, correspond to each PCI slot at any given time.

The p2q_flush signal is asserted at the rising edge of the first PCI clock (CLK) cycle after a config write (wr_cfg) cycle occurs or after an I/O write (iowr) cycle occurs or a memory write (memwr) cycle hits a downstream target (hit_tmem) during a command check state (cmd_chk_st). Gates 2014, 2016, 2018, and 2020, and flip-flop 2022 are arranged to produce p2q_flush in this way.

In the upstream bridge chip (i.e., when the upstream_chip_i signal is asserted), p2q_slot[2:0] always has a value of "001" since the CPU is the only master on the primary PCI bus. In the downstream chip, the value of p2q_slot depends upon whether the cycle leading to a flush condition is a cycle from the secondary bus 32 to the queue block 127 (i.e., if p2q_qcyc is asserted). If the p2q_qcyc signal is asserted, p2q_slot[2:0] takes on the value of the req_slot [2:0] signal produced by the PCI slave 121. The req_slot[2:0] signal indicates which of the seven devices on the secondary PCI bus 32 has been granted control of the bus 32. The PCI slave 121 generates the req_slot[2:0] signal by latching the value of the GNT# line for each of the seven slots on the bus 32 to form a seven bit latched grant signal (latched_gnt_[7:1]; the eighth grant line, which belongs to the queue block, is ignored) and encoding latched_gnt[7:1] according to look-up table 2006, as follows.

Creation of req_slot[2:0]	
latched_gnt_[7:1]	req_slot[2:0]
1111111	000
1111110	001
1111101	010
1111011	011
1110111	100
1101111	101
1011111	110
0111111	111

If the cycle leading to the flush is not a secondary-PCI-to-queue-block cycle, it must be an I/O read or config read to the target memory range of one of the slots on the secondary bus 32. When the cycle is an I/O read or config read (i.e., !iowr AND !wr_cfg), p2q_slot[2:0] takes on the value of the PCI slot whose memory range has been hit (mrange_slot[2:0]). Otherwise, the cycle is an I/O write or a config write, and p2q_slot[2:0] is set equal to "000" so that

of the PMWQ. For each transaction in the PMWQ, the tag memory 2036 maintains information such as the address to be written to, the PCI command code (MW or MWI), an address parity bit, and "locked cycle" and "dual address cycle" indication bits, as shown in the following table. The tag memory 2036 also stores a pointer to the data RAM location of the data corresponding to each of the transactions in the PMWQ.

Contents of PMWQ		
Field	**Bits**	**Comments**
Address	64	Upstream Transactions support Dual Address Cycles
PCI Command	1	Memory Write 0111 Memory Write and Invalidate 1111 (only necessary to store cbe [3])
Byte Enables	0	Store BEs on every valid transfer clock in the data RAM.
Parity	1/address 0	Must store PAR with each transfer along with 32-bit addr/data. Must store data parity bits on every valid data transfer in data RAM.
Data	0	Stored in data RAM up to 8 cache lines
Lock	1	
DAC Indication	1	Indicates whether address is 32 or 64 bits

Because the PCI Spec 2.1 requires posted memory write transactions to be executed in the order in which they are received, the tag memory 2036 is a circular FIFO device. The PMWQ, and therefore the tag memory 2036, can handle up to four posted memory write transactions simultaneously.

The data RAM 2038 includes four data buffers 2042, 2044, 2046, and 2048, one for each transaction in the PMWQ. Each buffer can store up to eight cache lines, or 256 bytes, of data (eight words per cache line). For each cache line in a buffer, the buffer stores eight data parity bits 2040 (one per dword) and thirty-two enable bits 2050 (one per byte).

A cable interface block 2060 receives each transaction and the corresponding data from the cable decoder and places the transaction in the tag memory 2036. A queue interface block 2053 receives the data from the cable interface block 2060 and places it in the appropriate location in the data RAM 2038. The queue interface 2053 also retrieves data from the data RAM 2038 and provides it to the QPIF when the QPIF is running the corresponding transaction on the PCI bus. An input pointer logic block 2054 generates four input pointers, one for each buffer, that tell the queue interface 2053 where to place the next word of data. A valid (output) pointer block 2056 generates four output pointers, one for each buffer, that indicate the position of the next word to be taken.

Referring also to Figure 60, a valid flag logic block 2052 maintains an eight bit valid line register 2062 for each of the four buffers in the data RAM 2038. The valid line register 2062 indicates which of the eight cache lines in each buffer contain valid data. When the last word in a cache line has been filled with data (i.e., valid_pointer[2:0] equals "111" and cd_next_data is asserted, indicating that the word has been filled), the corresponding bit in an eight bit cable valid signal (i.e., q0_cable_valid[7:0], q1_cable_valid[7:0], etc.) is set. The bit to be set is determined by the three most significant bits of the valid pointer (valid_pointer[5:3]), which indicate the cache line being filled. The corresponding bit in the cable valid signal also is set when a slot validation signal (validate_slot) is received from the cable decoder at the end of a transaction. The cable valid signal is latched into the valid line register 2062 corresponding to the selected data buffer at the rising edge of the first PCI clock cycle (CLK) after the last word is filled or the validate_slot signal is received. Otherwise, the valid line register maintains its current value. The bits in the valid line registers 2062 are cleared when the corresponding bits of an eight bit invalidate signal (i.e., q0_invalid[7:0], q1_invalid[7:0], etc.) is asserted.

The valid flag logic block 2052 generates a pmwq_valid[3:0] signal that indicates which, if any, of the four data buffers contains at least one valid line of data. The valid block 2052 also generates a pmwq_valid_lines[7:0] signal that indicates which of the eight cache lines of a selected data buffer are valid. A queue select signal from the QPIF (q2pif_queue_select[1:0]) is used to select which data buffer's valid line register 2062 is used to generate the pmwq_valid_lines[7:0] signal. When the queue block gains control of the bus to run a posted memory write cycle from a selected data buffer, the queue block transfers all data in each line whose corresponding bit is set in the pmwq_valid_lines[7:0] signal. Gates 2064, 2066, 2068, 2070, and 2072, and flip-flop 2074 are arranged to set the values in the valid line register 2062 for the first data buffer (q0_valid[7:0]). Similar circuitry determines the contents of the valid registers for the other three data buffers. Multiplexer 2076 selects the value of the pmwq_valid_lines[7:0] signal.

Referring now to Figure 61, a full line logic block 2058 maintains an eight bit full line register 2078 for each of the four data buffers. The contents of each full line register 2078 indicate which of the eight cache lines in the corresponding

(continued)

Contents of DRQ		
Field	Bits	Comments
PCI Command	4	I/O Read I/O Write Config Read Config Write Memory Read Memory Read Line Memory Read Multiple
Byte Enables	4	Byte Enables not necessary on MRL, MRM
Parity	1/address 1/data transfer	Send data par with delayed write transactions
Data	32	Data queued on delayed write transactions.
Lock	1	
DAC Indication	1	Indicates whether address is 32 or 64 bits
Buff Num	3	Indicates DCQ buffer allocated for completion data

Referring also to Figure 65, a valid flag logic block 2104 determines when the DRQ has received all of the information necessary to run the transactions in the queue memory 2100. When one of the DRQ slots is selected by a corresponding slot select signal (i.e., select_ zero for the first slot, select_one for the second slot, and select_two for the third slot) and the slot is validated by a validate_slot signal, indicating that the cable decoder has finished delivering the transaction to the DRQ, a valid signal corresponding to the slot (i.e., q0_valid, q1_valid, or q2_valid) is asserted at the rising edge of the next PCI clock (CLK) cycle. If a slot is not selected and validated by the validate_ slot signal, the slot's valid signal is deasserted if the QPIF has selected the slot by asserting a DRQ select signal (q2pif_drq_select) and identifying the slot (q2pif_queue_select = slot number) but has aborted the transaction by asserting a cycle abort signal (q2pif_abort_cycle). The valid signal also is deasserted if the DRQ ends the transaction by asserting a cycle complete signal (e.g., q0_cycle_complete) while the QPIF is waiting for more data (i.e., q2pif_next_data is asserted). However, the cycle complete signal is ignored if the QPIF is currently streaming data to the other bridge chip (i.e., q2pif_streaming is asserted). Otherwise, if the slot's valid signal is not specifically asserted or deasserted on a clock cycle, it retains its current value. The valid flag logic block 2104 also generates a DRQ valid signal (drq_valid[3:0]) that indicates which, if any, of the three DRQ slots contains a valid transaction, by combining the valid signals for each individual slot (i.e., drq_valid = {0, q2_valid, q1_valid, q0_valid}). Gates 2106, 2108, 2110, 2112, and 2114, multiplexers 2116 and 2118, and flip-flop 2120 are arranged to generate the slot valid signals and the DRQ valid signals in this manner.

The DRQ also includes pointer logic blocks that maintain pointers to the memory locations from which data is to be read during a delayed read request transactions. When the address at which the delayed read transaction will begin is loaded into the queue memory 2100, a valid pointer logic block 2122 generates a six bit valid pointer that indicates where the transaction will end. If the transaction involves a single word (e.g., a memory read), the valid pointer logic 2122 sets the valid pointer equal to the address loaded into the queue memory 2100. For a memory read line transaction, the valid pointer logic 2122 gives the valid pointer a value of "000111", which indicates that the last valid piece of data is eight dwords (i.e., one cache line) beyond the starting point. For a memory read multiple transaction, the valid pointer is set to "111111", which indicates that the last valid piece of data is sixty-four dwords (i.e., eight cache lines) beyond the starting point. The valid pointer logic 2122 maintains one valid pointer for each slot in the DRQ (valid_pointer_0[5:0], valid_pointer_1[5:0], and valid_pointer_2[5:0]). The location of the valid pointer is ignored by the DRQ when it receives a streaming signal from the QPIF (q2pif_streaming), as described in more detail below.

An output pointer logic block 2124 maintains three output pointers (output_pointer_0[5:0], output_pointer_1[5:0], and output_pointer_2[5:0]), one for each slot in the DRQ, that indicate the next word of data to be read from memory and delivered to the other bridge chip. The pointer is incremented when the QPIF indicates that it is ready to read the next piece of data (i.e., it asserts the q2pif_next_data signal), once for every word read. Except in streaming situations, a transaction is terminated (completed) when the output pointer reaches the valid pointer. If a transaction terminates before all of the data is read (i.e., before the output pointer reaches the input pointer), the QPIF will pick up at the location indicated by the output pointer when the transaction resumes. If the output pointer is incremented but the

reach the requesting device because the transaction was terminated by a device other than the QPIF, the QPIF asserts a stepback signal (q2pif_step_back) that causes the output pointer logic block 2140 to decrement the output pointer by one word.

The third pointer block, a valid pointer logic block 2142, maintains for each of the eight data buffers a six bit valid pointer (valid_pointer_0[5:0], valid_pointer_1[5:0], etc.) that indicates the next word of data in the corresponding data buffer that is available to the QPIF. Because the PCI Spec 2.1 requires that read completion data not be returned before an earlier-initiated posted memory write transaction, delayed completion data placed into the DCQ while a posted memory write is pending in the PMWQ cannot be made available to the requesting device until the posted memory write is completed on the PCI bus and removed from the PMWQ. Therefore, as long as any earlier-enqueued posted memory write transactions remain in the PMWQ, the valid pointer must remain at its current position. Then, when all earlier-enqueued posted memory writes have been removed from the PMWQ, the valid pointer may be moved to the same position as the in pointer. When the PMWQ is empty, all delayed completion data is valid (i.e., available to the requesting device) as soon as it is stored in the DCQ.

Referring also to Figures 67A and 67B, the valid pointer logic block 2142 must ask the master cycle arbiter (MCA) to validate all delayed completion transactions that enter the delayed completion queue while a posted memory write is pending in the PMWQ. But because the MCA can enqueue no more than four delayed completion transactions at once, as discussed below, the valid pointer logic block 2142 may request validation of no more than four delayed completion data buffers at once. The valid pointer logic block 2142 also must keep track of which four delayed completions transactions are enqueued in the MCA at any given time. To do so, the valid pointer logic block 2142 maintains two four-slot registers: a DCQ buffer number register 2144 and a validation request register 2146. The buffer number register 2144 maintains the three-bit DCQ buffer number, as determined by the DCQ buffer number signal (cd_dcq_buff_num[2:0]) provided by the cable decoder, of each delayed completion transaction enqueued in the MCA. The validation request register 2146 maintains one transaction validation request bit for each of the DCQ buffers whose numbers are stored in the four slots 2148a-d of the buffer number register 2144. The request bit in each slot 2150a-d of the validation request register 2146 is asserted if a corresponding delayed completion transaction is enqueued in the MCA. The values of the bits in the four validation request slots 2150a-d are provided together to the MCA as a four bit validation request signal (dcq_valid[3:0]).

When a delayed completion transaction is to be enqueued in the MCA, its corresponding DCQ buffer number is loaded into one of the buffer number slots 2148a-d by the cd_dcq_buff_num[2:0] signal. The slot 2148a-d to be loaded is selected by a two bit selection signal (next_valid_select[1:0]). The value of the selection signal depends upon the value of the dcq_valid[3:0] signal generated by the validation request register 2146 and look-up table 2152, the contents of which are shown in the table below. The slot is loaded when it is selected by next_valid_select[1:0], when the cable decoder has selected the DCQ and has completed the transaction (i.e., cd_dcq_select and cd_complete are asserted), and when at least one posted memory write transaction is pending in the PMWQ (i.e., pmwq_no_pmw is not asserted). Gates 2154, 2156, 2158, 2160, and 2162 and 2x4 decoder 2164 are arranged to load the buffer number register 2144 in this manner. Likewise, the corresponding bit in the validation request register 2146 is set by the output of gates 2154, 2156, 2158, 2160, and 2162 and 2x4 decoder 2164.

Buffer number register slot selection		
dcq_valid[3:0]	**next_valid_select[1:0]**	**slot #**
xxx0	00	0
xx01	01	1
x011	10	2
0111	11	3

In response to the dcq_valid[3:0] signal, the MCA outputs a four bit DCQ run signal (mca_run_dcq[3:0]) that indicates which of the DCQ buffers pointed to by the buffer number register may have its valid pointer updated. The mca_run_dcq[3:0] signal is provided to a valid pointer update logic block 2166, along with the pmwq_no_pmw signal and the in pointers for each of the eight data buffers. If a posted memory write transaction remains in the PMWQ after the MCA asserts one of the mca_run_dcq[3:0] bits (which will happen when a posted memory write transaction was enqueued after the delayed completion transaction was enqueued but before the MCA asserted the corresponding mca_run_dcq bit), the corresponding valid pointer is updated as long as no other delayed completion transactions corresponding to the same DCQ buffer are still enqueued in the MCA. If a delayed completion transaction for the same DCQ buffer is still enqueued in the MCA, the valid pointer may be updated only when the mca_run_dcq bit corresponding this transaction is asserted. On the other hand, as soon as the pmwq_no_pmw signal is deasserted, all valid pointers

CPU), this feature is disabled in the upstream chip. Gate 2190 and multiplexer 2192 are arranged to generate the q2a_stream and q2a_stream_master signals.

When a requesting device hits a delayed completion message stored in the DCQ, the corresponding bit of an eight bit hit signal (hit[7:0]) is asserted. The hit[7:0] signal indicates which of the eight DCQ buffers was hit by the current request. When this happens, if the corresponding DCQ buffer contains data (i.e., dcq_no_data is not asserted), the stream logic 2180 latches the value of the hit signal for the duration of the transaction (i.e., as long as q2pif_cyc_complete is asserted). The latched version of the hit signal forms a "delayed" hit signal (dly_hit[7:0]). When either the hit signal or the delayed hit signal indicates that a DCQ buffer has been hit, a three bit DCQ stream buffer signal (dcq_stream_buff[2:0]) provides the buffer number of the hit DCQ buffer. Then, if the cable decoder places delayed completion data into the buffer while the cycle that hit the buffer is in progress (i.e., cd_dcq_select is asserted and cd_dcq_buff_num[2:0] equals dcq_stream_buff[2:0]), the stream logic block 2180 asserts a stream connect signal (dcq_stream_connect) that tells the QPIF that a stream has been established. The QPIF then informs the bridge chip on the target bus that a stream has been established. If certain conditions are met, the target QPIF will continue to stream until it is told to stop by the initiating QPIF, as discussed in more detail below. Gates 2194 and 2196, multiplexers 2198 and 2200, and flip-flop 2202 are arranged to generate the delayed hit signal. Gates 2204, 2206, and 2208 and encoder 2210 are arranged as shown to generate the dcq_stream_connect and dcq_stream_buff[2:0] signals.

Referring again to Figure 66, the DCQ will, under certain circumstances, automatically prefetch data from the target bus on behalf of a PCI master in anticipation that the master will come back and request the data. A prefetch logic block 2212 in the DCQ prefetches data when the reading master consumes all of the data in its DCQ buffer and the prefetch logic 2212 anticipates that the requesting device will return with a sequential read request (i.e., a request that picks up with data located at the next sequential location in memory). Because some devices, such as multi-threaded masters, routinely read all of the data requested in one transaction and then return with a different, non-sequential request, the prefetch logic 2212 includes prediction circuitry that disables the prefetch capabilities for each device on the PCI bus until the device has shown a tendency to issue sequential read requests. As soon as a device that has been receiving prefetched data returns with a non-sequential read request, the prediction circuitry will disable the prefetching function for that master.

Referring also to Figures 69A and 69B, the prefetch logic block 2212 includes a prefetch prediction register 2214, the output of which is an eight bit prefetch enable signal (prefetch_set[7:0]) that governs whether the prefetch function is available for each of the devices on the PCI bus. All bits in the prefetch enable signal are cleared at reset (RST) and when the QPIF orders a general flush of all of the DCQ registers (i.e., general_flush is asserted and q2pif_slot[2:0] equals "000"). The general_flush signal is discussed in more detail below. Gates 2216 and 2218 generate the signal that resets the prefetch_set bits.

An individual bit in the prefetch enable signal is set when the corresponding PCI slot is selected by the q2pif_slot signal and the following conditions occur: the requesting device hits a delayed completion buffer in the DCQ (i.e., one of the bits in the cycle_hit[7:0] signal is asserted), the current transaction is a memory read line or memory read multiple cycle (i.e., q2pif_cmd[3:0] equals "1100" or "11110"), the QPIF has indicated that the cycle is complete (i.e., q2pif_cyc_complete is asserted), and the last word of data was taken from the DCQ buffer (i.e., last_word is asserted). Gates 2220, 2222, 2224 and 2228a-h and decoder 2226 are arranged to set the prediction bits in this manner. The last_word signal is asserted by the prefetch logic 2212 when the requesting device tries to read past the end of the DCQ buffer. This occurs when the out pointer and in pointer are equal, indicating that the end of the DCQ buffer has been reached (i.e., for a four cache line buffer, out_pointer_x[4:0] equals valid_pointer_x[4:0] or, for an eight cache line buffer, out_pointer_x[5:0] equals valid_pointer_x[5:0]) and when the requesting device tries to read another piece of data (i.e., q2pif_next_data is asserted). Gates 2230, 2232, and 2234 are arranged to generate the last_word signal.

An individual bit in the prefetch enable signal is cleared when the corresponding PCI slot is selected and either a PCI flush condition occurs (p2q_flush is asserted), the QPIF tells the DCQ to step back the buffer's valid pointer (q2p_step_back is asserted), or the requesting device initiates a transaction that misses all of the DCQ buffers (q2pif_check_cyc is asserted and dcq_hit is deasserted). Gates 2236, 2238, and 2240a-h and decoder 2226 are arranged to clear the prediction enable bits in this manner.

When the prefetching function is enabled for a device on the PCI bus, the prefetch logic 2212 can generate two types of prefetch signals for the device: a prefetch line signal (dcq_prefetch_line) and a prefetch multiple signal (dcq_prefetch_mul). The prefetch line signal is generated when the current PCI command from the requesting device is a memory read line signal, and the prefetch multiple signal is generated when the current PCI command is a memory read multiple signal. In either case, the corresponding prefetch signal is generated when the following conditions occur: the prefetch_set bit for the requesting PCI slot is set; a corresponding prefetch enable bit in the configuration registers is set (cfg2q_auto_prefetch_enable); the DRQ in the upstream chip is not full (!tc_dc_full); the DCQ buffer has room for the corresponding amount of prefetch data (!dcq_no_prefetch_room); the current cycle hit the DCQ buffer; and the requesting master has tried to read past the end of the DCQ buffer (last_word and q2pif_cyc_complete). Gates 2242, 2244, 2246, 2248, 2250, and 2252, decoder 2254, and multiplexers 2256 and 2258 are arranged to generate the

When a device on the PCI bus initiates a delayed read request and a DCQ completion buffer is set aside, the buffer state logic 2264 changes the buffer's state to PartComplete. If the DCQ initiates a prefetch read, the buffer state is changed to PartPrefetch. When the last piece of completion data arrives, the buffer's state is changed from PartComplete or PartPrefetch to Complete or Prefetch, respectively. When the requesting device resubmits a retried read request and hits the buffer, any valid completion data is given to the master if the buffer is in the Complete, Prefetch, PartComplete, or PartPrefetch state. If the master does not take all of the data before disconnecting, the buffer's state is changed to Prefetch or PartPrefetch to indicate that the unclaimed data is considered to be prefetch data. If the master takes the last piece of data when the buffer is in the Complete or Prefetch state, the buffer's state is changed to Empty.

If a flush signal is received while a buffer is in the Prefetch state, the prefetch data in the buffer is discarded and the buffer state is changed to Empty. If a flush event occurs while the buffer is in the PartPrefetch state and completion data is still arriving, the buffer is changed to the Discard state until all of the prefetch data arrives. When the transaction is complete, the prefetch data is discarded and the buffer state is changed to Empty. If the buffer is in the Complete or PartComplete state when a flush signal is received, the completion data is left in the buffer and the buffer state remains unchanged. If the flush signal occurs because the corresponding PCI device has issued a new request (i.e., a request that is not currently enqueued and that "misses" all of the completion buffers), as discussed below, the DCQ allocates a new buffer for the transaction, as discussed above. Therefore, a PCI device may have more than one completion buffer allocated. Multiple buffers may be allocated to a PCI device when the device has a buffer containing or awaiting completion data (i.e., the buffer is in the Complete or PartComplete state) and the device issues a new request. Because multi-threaded devices are the only devices that can maintain multiple transactions at once, only multi-threaded devices can have multiple completion buffers reserved simultaneously.

MASTER CYCLE ARBITER

The Master Cycle Arbiter (MCA) determines the execution order of posted memory write and delayed request transactions while maintaining the ordering constraints between posted memory write, delayed request, and delayed completion cycles set forth in the PCI Spec 2.1. According to the PCI Spec 2.1, the MCA must guarantee that executed cycles maintain strong write ordering and that no deadlocks occur. To ensure that no deadlocks will occur, posted memory write cycles must be allowed to pass earlier enqueued delayed request cycles, and to maintain the required ordering constraints, delayed request cycles and delayed completion cycles must never be allowed to pass earlier-enqueued posted memory write cycles.

Referring to Figure 70, the MCA uses two transaction queues, a transaction run queue (TRQ) (or transaction execution queue) 2270 and a transaction order queue (TOQ) 2272, to manage cycles enqueued in the PMWQ, DRQ, and DCQ. An MCA control block 2274 receives transactions from the PMWQ, DRQ, and DCQ in the form of four bit validation request signals (pmwq_valid[3:0], drq_valid[3:0], and dcq_valid[3:0]) and outputs run commands in the form of four bit run signals (mca_run_pmwq[3:0], mca_run_drq[3:0], and mca_run_dcq[3:0]). The transactions are moved into and out of the TRQ 2270 and TOQ 2272 by a TRQ control block 2276 and a TOQ control block 2278, respectively.

Referring also to Figure 71, the TRQ 2270 is the queue from which the MCA determines the transaction execution order. Transactions in the TRQ 2270 can be executed in any order without violating the transaction ordering rules, but once a posted memory write cycle is placed in the TRQ 2270, no other cycle can be placed in the TRQ 2270 until the posted memory write is removed. Transactions in the TRQ 270 are tried in circular order and generally are completed in the order they were. received. However, if a transaction in the TRQ 2270 is retried on the PCI bus, the MCA may select the next transaction in the TRQ 2270 to be tried on the PCI bus. Because delayed completion transactions are slave cycles rather than master cycles, they are never placed in the TRQ 2270. Furthermore, because delayed completion information may be made available to the requesting device as soon as it enters the DCQ if no posted memory write cycles are pending in the PMWQ, delayed completion transactions are placed in the TOQ 2272 only when a posted memory write cycle is pending in the TRQ 2270, as discussed in more detail below.

The TRQ 2270 is a circular queue that holds up to four transactions at once. Because the MCA must always be able to run at least one posted memory write transaction to preserve the required ordering constraints, the TRQ 2270 can never hold more than three delayed request transactions at once. Furthermore the TRQ can hold only one posted write transaction at a time because posted writes cannot be passed by any later-initiated transaction, including other posted writes. Each slot 2280a-d in the TRQ 2270 contains three bits of information: a one bit cycle type indicator 2282 (which equals "1" for posted memory write transactions and "0" for delayed request transactions), and a two bit valid pointer 2284, the four possible values of which identify which of the buffers in the PMWQ or the DRQ the enqueued transactions occupy. The TRQ 2270 also includes an input/output enable block 2286 that determines when a transaction may be moved into or out of the TRQ 2270, an input logic block 2288 that controls the placement of a transaction into the TRQ 2270, and an output logic block 2290 that controls removal of a transaction from the TRQ 2270. These logic blocks contain standard queue management circuitry.

A circular input pointer 2292 selects the next available slot for placement of an incoming transaction. The input

When a delayed request transaction or posted memory write transaction is popped out of the TOQ 2272, the transaction is placed in the TRQ 2270 to await arbitration. But because delayed completion transactions are target transactions and not master transactions, delayed completions are not placed in the TRQ 2270. Instead, delayed completions are simply popped out of the TOQ 2272 and used to validate the corresponding data in the DCQ data buffers. However, as long as a posted memory write transaction is enqueued in the TRQ 2270, all delayed completions must be placed in the TOQ 2272, even when two or more delayed completions correspond to the same delayed request and therefore the same delayed completion buffer, as described above.

Referring to Figures 73A through 73D, the MCA control block 2274 controls the flow of transactions through the MCA. As discussed above, the PMWQ, DRQ, and DCQ request validation of transactions held in the queues by providing four bit validation signals pmwq_valid[3:0], drq_valid[3:0], and dcq_valid[3:0], respectively, to the MCA. Among these signals, only one bit can change during each clock pulse since only a single new transaction can be placed into the queue block on each clock pulse. Therefore, the MCA control block identifies new validation requests by watching for the changing bits in the pmwq_valid, drq_valid, and dcq_valid signals. To do so, the MCA control block latches and inverts each signal at the rising edge of every PCI clock to create a delayed, inverted signal and compares the delayed, inverted signal to the current signal (i.e., the signal at the next clock pulse). Since only a newly changed bit will have the same value as its delayed and inverted counterpart, the MCA control block is able to detect which bit changed. Using flip-flops 2340, 2342, and 2344 and gates 2346, 2348, and 2350, the MCA controller generates new_pmwq_valid [3:0], new_drq_valid[3:0], and new_dcq_valid[3:0] signals which, at each clock pulse, together identify whether the PMWQ, DRQ, or DCQ, if any, submitted a new transaction for validation and which buffer in the corresponding queue contains the new transaction. Referring also to Figure 74, the MCA control block uses a look-up table 2352 to convert the twelve bits of the new_pmwq_valid, new_drq_valid, and new_dcq_valid signals into the two bit d_valid[1:0] and d_cyctype[1:0] signals provided to the TRQ and TOQ, as discussed above.

The MCA controller enables the TOQ by latching the toq_enabled signal to a value of "1" when either the trq_pmw is asserted, indicating that a posted memory write cycle is enqueued in the TRQ, or when the toq_enable signal already is asserted and the TOQ is not empty (!toq_empty). Gates 2354 and 2356 and flip-flop 2358 are arranged to generate toq_enabled in this manner.

The MCA control block asserts the new_toq_cycle signal, which instructs the TRQ to enqueue the cycle being popped out of the TOQ, when there was not a posted memory write cycle in the TRQ during the previous clock cycle (!s1_trq_pmw), when the TOQ is not empty (!toq_empty), and when the cycle being popped out of the TOQ is not a delayed completion transaction (!(toq_cyctypeo[1] = "DC")). The MCA controller uses gate 2360 to generate the new_toq_cycle signal.

The next_toq_cycle signal, which is used to increment the TOQ output counter to the next cycle in the TOQ, is asserted when the TOQ is not empty (!toq_empty) and either when no posted memory write cycles currently are enqueued in the TRQ (!trq_pmw) and the next cycle in the TOQ is a delayed completion (toq_cyctype[1] = "DC") or when the next TOQ cycle is a posted memory write or delayed request transaction (!(toq_cyctype[1] = "DC")) and there were no posted memory write transactions during the previous clock cycle (!s1_trq_pmw). The control block uses gates 2362, 2364, 2366, and 2368 to generate the next_toq_cycle signal.

The MCA controller generates the mca_run_dcq[3:0] signal to indicate that a delayed completion transaction has been popped out of the TOQ. When the TRQ contains no posted memory write cycles (!trq_pmw), the TOQ is not empty (!toq_empty), and the TOQ cycle is a delayed completion (toq_cyctype [1] = "DC"), the mca_run_dcq[3:0] signal takes on the value of the decoded toq_valido[1:0] signal, discussed above. Otherwise, the mca_run_dcq[3:0] signal equals "0000". Gate 2370, decoder 2372, and multiplexer 2374 are arranged to generate mca mca_run_dcq[3:0] in this manner.

The MCA control block generates new_mca_run_dr[3:0] and new_mca_run_pmw[3:0] signals to indicate that it has a new delayed request transaction and a posted memory write transaction, respectively, to be enqueued. The new_mca_run_dr[3:0] signal takes on the value of the 2x4 decoded d_valido[1:0] signal, discussed above, when the new cycle is a delayed request cycle (d_cyctype[0] = "DR"). Otherwise, all bits of the new_mca_run_dr signal are set to zero. Likewise, the new_mca_run_pmw[3:0] signal takes on the value of the 2x4 decoded d_valido[1:0] signal when the new cycle is a posted memory write transaction and is set to "0000" otherwise. Decoders 2376 and 2380 and multiplexers 2378 and 2382 are arranged to generate the new_mca_run_dr and new_mca_run_pmw signals in this manner.

The MCA controller generates toq_mca_run_dr[3:0] and toq_mca_run_pmw[3:0] signals to indicate when a new delayed request transaction or posted memory write transaction, respectively, has popped out of the TOQ. The toq_mca_run_dr[3:0] signal takes on the value of the 2x4 decoded toq_valido[1:0] signal when a delayed request cycle is popped out of the TOQ and a value of "0000" otherwise. Likewise, the toq_mca_run_pmw[3:0] signal takes on the value of the 2x4 decoded toq_valido[1:0] signal when a posted memory write cycle pops out of the TOQ and a value of "0000" otherwise. Decoders 2384 and 2388 and multiplexers 2386 and 2390 are used to generate the toq_mca_run_dr and toq_mca_run_pmw signals in this manner.

signal from the PCI interface (p2q_ad[31:2]) is loaded into the address register 2512. The address register 2512 outputs the address signal used by the QPIF (q2pif_addr[31:2]). The second register is a four bit command register 2514 that receives the PCI command code from the PCI bus (p2q_cmd[3:0]) and outputs the QPIF command signal (q2pif_cmd [3:0]). The third register is a three bit slot selection register 2516 that receives the p2q_slot[2:0] signal indicating which PCI device is the current bus master and outputs the QPIF slot selection signal (q2pif_slot[2:0]).

When the address phase of the PCI transaction ends, the slave state machine 2502 asserts a data phase latching signal (reg_latch_second_request) indicating that the data phase information should be latched from the PCI bus. At the next falling edge of the PCI clock signal, the asserted reg_latch_first_request signal causes a delayed data phase latching signal (dly_reg_latch_second_request) to be asserted. When both the original and the delayed data phase latching signals are asserted, the latching logic 2506 generates a second latching signal (latch2). Flip-flop 2518 and gate 2520 are arranged to generate the second latching signal in this manner.

The latching logic 2506 then loads the data phase information from the PCI bus (via the PCI interface) into three data phase registers when the second latching signal is asserted. The first data phase register is a thirty-two bit data register 2522 that receives the data associated with the current transaction on the PCI address/data lines (p2q_ad[31:0]) and outputs the QPIF data signal (q2pif_data[31:0]). The second data phase register is a four bit enable register 2524 that receives enable bits from the PCI bus (p2q_cbe[3:0]) and outputs the QPIF byte enable signal (q2pif_byte_en [3:0]). The third register is a three bit lock register 2526 that receives the PCI lock signal (p2q_lock) indicating that the current transaction should be run as a locked transaction and outputs the QPIF lock signal (q2pif_lock).

Referring again to Figure 75 and also to Figure 77, the QPIF includes a "lock" logic block 2528 that controls the "lock" state of the QPIF. The QPIF has three lock states: an unlocked state 2530 (lock_state[1:0] = "00") that indicates that no locked transactions are pending in the DCQ; a locked state 2532 (lock_state[1:0] = "01) indicating that a locked transaction has been received in the DCQ or is completing on the PCI bus; and an unlocked-but-retry state 2534 (lock_state[1:0] = "10") that indicates that the lock has been removed but that a posted memory write transaction pending in the other bridge chip must be run before the next transaction can be accepted.

At power-up and reset, the lock logic 2528 enters the unlocked state 2530 and waits for a locked transaction to enter the DCQ (indicated by the assertion of the dcq_locked signal). At the first clock pulse after the dcq_locked signal is asserted, the lock logic enters the locked state 2532, which forces the QPIF slave state machine 2502 to retry all transaction requests from the PCI bus. The PCI interface also asserts a lock signal (p2q_lock) that indicates it has locked the PCI bus for the transaction. After the locked transaction has completed and the requesting device has retrieved the locked completion data from the DCQ, the dcq_locked signal is deasserted. At the first clock pulse after the dcq_locked is deasserted, while the p2q_lock signal is still asserted, if no posted memory writes are pending in the other bridge chip (i.e., the pmw_empty signal is asserted by the cable decoder), the lock logic 2528 returns to the unlocked state 2530 and the slave state machine 2502 again is able to accept transaction requests. However, if the pwm_empty signal is not asserted at the first clock pulse after the dcq_lock signal is deasserted, the lock logic 2528 enters the unlocked-but-retry state 2534, which forces the slave state machine 2502 to retry all transactions until the posted memory write cycle is completed on the other PCI bus. After the posted memory write cycle is complete, the pwm_empty signal is asserted, and the lock logic 2528 returns to the unlocked state 2530.

Referring again to Figure 75 and also to Figure 78, the QPIF includes buffer flush logic 2536 that determines when the DCQ should flush data from one or all of its data buffers. As discussed above, the PCI interface in the downstream chip generates a p2q_flush signal when the upstream chip issues an I/O or config write or a memory write that hits the target memory range register (TMRR) of a downstream device. The QPIF buffer flush logic 2536 asserts a QPIF flush signal (general_flush) that flushes the corresponding data buffer or all data buffers (depending upon the value of the p2q_slot signal, as discussed above) when the p2q_flush signal is received. Otherwise, the buffer flush logic 2536 asserts the general flush signal only when a device on the secondary bus issues a delayed request that misses all of the DCQ buffers when checked by the DCQ control logic (i.e., !dcq_hit and q2pif_check_cyc are asserted). In either case, the general_flush signal is used to flush only buffers that are in the "prefetch" state, as discussed above. Therefore, prefetch data is held in the DCQ until the PCI interface orders a flush or until the corresponding PCI device issues a non-sequential request (i.e., misses the DCQ). Gates 2538 and 2540 are arranged to generate the general_flush signal in this manner.

When a multi-threaded device has more than one completion buffer allocated, at least one of which contains prefetch data, the prefetch data remains in the corresponding buffer as long as the device does not issue a request that misses all of the DCQ buffers. As soon as the device issues a new request, all of its prefetch buffers are flushed. Alternatively, a prefetch buffer associated with a multi-threaded device could be flushed as soon as the device issues a request that hits another DCQ buffer.

Referring again to Figure 75, the QPIF includes a read command logic block 2542 that receives read commands from the PCI interface and prefetch commands from the DCQ and provides an outgoing message command signal (message_cmd) to the cable. In non-streaming situations, the outgoing message command may be same as the command received from the PCI bus or the DCQ, or the read command logic 2542 may convert the command into one

Referring to Figures 81A through 81C, the write command logic block 2566 generates a four bit write command signal (write_cmd[3:0]) indicating the command code of the posted write transaction to be executed on the PCI bus. If the command code stored in the PMWQ represents a memory write and invalidate command (pmwq_cmd[3] = "1"), the write command logic 2566 generates a write command code of "1111". If the PMWQ command code represents a memory write command, the write command logic 2566 looks at the memory-write-to-memory-write-and-invalidate configuration bit (cfg2q_mw2mwi) corresponding to the target PCI slot. If the cfg2q_mw2mwi bit is not set, the write command logic 2566 produces a memory write command ("0111"). If the configuration bit is set, the write command logic 2566 generates a MWI command if the next line in the PMWQ data buffer is full (pmwq_full_line is asserted) and generates a MW command otherwise. Multiplexers 2568 and 2570 are arranged to generate the write_cmd signal in this manner.

When the QPIF is executing a transaction on the PCI bus and has reached a cache line boundary, the write command logic 2566 may assert a new_write_cmd signal indicating that the current transaction must be terminated in favor of a new write command. If the transaction has reached the last cache line in the PMWQ data buffer (i.e., pmwq_pointer [5:3] equals "111"), the new_write command signal is asserted to indicate that the transaction should be terminated if the next PMWQ buffer is not an overflow buffer containing valid data, if the corresponding cfg2q_mw2mwi bit is not set, or if the full_line bits corresponding to the current cache line and the next cache line are different (i.e., pmwq_full_line [7] does not equal pmwq_next_full_line). If the transaction has not reached the end of the PMWQ buffer, the new_write_cmd signal is asserted either if the next line in the PMWQ buffer does not contain valid data (! pmwq_valid_lines[x+1]) or if the cfg2q_mw2mwi bit is set and the full line bits for the current line and the next line are different (i.e., pmwq_full_line[x] does not equal pmwq_full_line[x+1]). Gates 2572, 2574, 2576, 2578, and 2580 and multiplexer 2582 are arranged to generate the new_write command signal in this manner.

After the new_write_cmd signal is asserted, the transaction is not terminated until the write command logic block 2566 asserts a synchronous new write command signal (held_new_write_cmd). The held_new_write_cmd signal is asserted at the first clock pulse after the new_write_cmd signal is asserted and the end_of_line signal is asserted indicating that the end of the cache line has been reached, as long as the PCI interface has not terminated the transaction (i.e., p2q_start_pulse is asserted). The held_new_write command is deasserted at reset and at the first clock pulse after the new_write_cmd, end_of_line, and p2q_start_pulse signals are deasserted and the QPIF terminates the transaction (i.e., the asynchronous early_cyc_complete signal is asserted). Otherwise, the held_new_write_cmd signal retains its current value. Gates 2584 and 2586, inverter 2588, and flip-flop 2590 are arranged to generate the held_new_write_cmd signal in this manner.

Referring again to Figure 75 and also to Figure 82A, the QPIF includes an overflow logic block 2600 that allows the master state machine 2500 to manage overflow data, if any, when executing a posted write transaction on the target bus. When the QPIF receives a transaction run signal (mca_run_pmw or mca_run_dr, discussed above) from the MCA, the overflow logic 2600 generates a two bit initial queue selection signal (start_queue_select[2:0]) indicating which of the buffers in the PMWQ or DRQ should be selected to run the current transaction. The following table shows how the start_queue_select signal is generated.

Creation of start_queue_select signal	
MCA Run Code {mca_run_pmw, mca_run_dr}	**start_queue_select**
00000001	00
00000010	01
00000100	10
00001000	11
00010000	00
00100000	01
01000000	10
10000000	11

When the QPIF is executing a posted write transaction on the target bus, a two bit QPIF queue selection signal (q2pif_queue_select[1:0]) is used to select the appropriate buffer in the PMWQ. When the transaction is initiated, the master state machine 2500 asserts a queue selection signal (initial_queue_select) that causes the q2pif_queue_select signal to take on the value of the initial queue selection signal (start_queue_select). At the same time, a queue selection counter 2602 is loaded with the value of the start_queue_select signal. After the initial_queue_select signal is deas-

<table>
<tr><th colspan="3">MASTER STATE MACHINE</th></tr>
<tr><th>Current State</th><th>Event</th><th>Next State</th></tr>
<tr><td>IDLE

IDLE
IDLE
IDLE</td><td>A=(any_run&&!cable_busy&&!p2q_master_dphase)
| (any_run_drg && tc_dc_full)
B: p2q_ack && q2p_dac_flag
C: p2q_ack && any_drg_run
D: p2q_ack && !(q2p_dac_flag | any_drq_run)</td><td>IDLE

MASTER_DAC
RDATA1
WDATA1</td></tr>
<tr><td>MASTER_DAC</td><td>E: p2q_ack && any_drq_run && p2q_start_pulse
F: p2q_ack && p2q_start pulse &&!any_drq_run
G: !p2q_ack</td><td>RDATA1
WDATA1
IDLE</td></tr>
<tr><td>RDATA1</td><td>H: !p2q_ack
I: p2q-ack && p2q_start_pulse
J: p2q_ack && !p2q_start-pulse</td><td>IDLE
RBURST
RDATA1</td></tr>
<tr><td>RBURST</td><td>K: !p2q_ack | p2q_retry | p2q_target_abort |
(queue_cyc_complete && !(!p2q_last_dphase &&
p2q_master_dphase && cd_stream && stream_match &&
!cfg2q_stream_disable) && !p2q_trdy) |
(read_page_disconnect && !p2q_trdy)
L: p2q_ack && !p2q_retry && !p2q_target_abort &&
((read_page_disconnect && p2q_trdy) | (queue_cyc_
complete && ((!p2q_last_dphase && p2q_master_dphase
&& cd_stream && stream_match && !cfa2q_stream_disable)
| p2q_trdy)) | !p2q_trdy | otherwise)</td><td>IDLE

RBURST</td></tr>
<tr><td>WDATA1</td><td>M: !p2q_ack | p2q_retry | p2q_target_abort |
((queue_cyc_complete | held_new_write_cmd |
end_of_line && new_write_cmd | p2q_last_dphase
| sl_p2q_last_dphase) && !p2q_trdy)
N: p2q_ack && !p2q_retry && !p2q_target_abort
&& (queue_cyc_complete | held_new_write_cmd |
end_of_line && new_write_cmd | p2q_last_dphase |
sl_p2q_last_dphase) && p2q_trdy
O: otherwise</td><td>IDLE

WDATA1

WDATA2</td></tr>
<tr><td>WDATA2</td><td>P: !p2q_ack | (p2q_retry&&!p2q_trdy) | p2q_target_abort
Q: p2q_ack && p2q_retry && p2q_trdy

R: p2q_ack && !p2q_retry && !p2q_target_abort &&
(queue_cyc_complete | end_of_line && new_write_
cmd) && (!p2q_trdy | p2q_start_pulse)
S: otherwise</td><td>IDLE
WRETRY

WSHORT_BURST

WDATA2</td></tr>
<tr><td>WRETRY</td><td>T: Always</td><td>IDLE</td></tr>
<tr><td>WSHORT_BURST</td><td>U: !p2q_ack | p2q_retry | p2q_target_abort
V: p2q_ack && !p2q_retry && !p2q_target_abort
&&((overflow_next_queue && !new_write_cmd
&& !p2q_trdy) | !p2q_trdy)
W: otherwise</td><td>IDLE
WCOMPLETE

WSHORT_BURST</td></tr>
</table>

Then, if the p2q_ack signal is deasserted, or if the QPIF transaction is retried by the PCI interface (p2q_retry is asserted), or if the PCI interface aborts the transaction (p2q_target_abort is asserted), the master state machine terminates the transaction on the PCI bus, aborts the completion message on the cable, and returns to the IDLE state. When the p2q_ack signal is taken away, the master cycle arbiter continues to select the current transaction. But when the transaction is retried or aborted, the master state machine steps the MCA to the next transaction.

While the p2q_ack signal is still asserted and the QPIF transaction is not retried or aborted, the master state machine nevertheless terminates the transaction and returns to the IDLE state 2700 if a 4K page boundary is reached and the PCI interface indicates that the target device has stopped taking data (p2q_trdy is no longer asserted). If the target device took the last piece of data, the master state machine remains in the RBURST state 2708.

If the QPIF asserts the queue_cyc_complete signal indicating that the transaction has completed, the master in general will terminate the transaction and return to the IDLE state 2700 if the p2q_trdy signal is deasserted or remain in the RBURST state 2708 until the last dword of data is transferred if the p2q_trdy signal remains asserted. However, if the transaction is in the data phase and is not in the last data phase (p2q_master_dphase and !p2q_last_dphase) and a stream has been established with the other bridge chip (cd_stream and stream_match and ! cfg2q_stream_disable), the master state machine will remain in the RBURST phase indefinitely. When the QPIF is streaming, the master state machine asserts a streaming signal (q2pif_streaming) that forces the QPIF to continue to provide data to the requesting device on the other PCI bus until that device terminates the transaction.

If the p2q_ack signal remains asserted and neither the p2q_retry, p2q_target_abort, or queue_cyc_complete signals are asserted, the master state machine looks at the p2q_trdy signal. If the signal is not asserted, indicating that the target device has taken or provided the last piece of data, the master state machine asserts its next data signal (early_next_data), which indicates that the QPIF is ready to transfer another piece of data. The next data signal is asserted only if the transaction is not a corrected read (align_read is not asserted) or if the transaction is a corrected read and the read_data_start signal has been asserted. If the p2q_trdy signal is asserted, indicating that the target has not performed the last data transfer, the state machine remains in the RBURST state 2708.

In the WDATA1 state 2706, the master state machine begins the data phase of a posted memory write transaction. If the p2q_ack signal is deasserted or the p2q_retry or p2q_target_abort signals are asserted while the master state machine is in this state, the transaction is terminated on the PCI bus and the state machine returns to the IDLE state 2700. When the p2q_ack signal is deasserted, the MCA remains on the current cycle; otherwise, the master state machine steps the MCA to the next transaction.

If the p2q_ack signal remains asserted and the transaction is neither retried nor aborted, the master state machine must determine whether the write involves a single dword or more than one dword. If in the WDATA1 state the queue_cyc_complete signal is asserted, the held new write command signal is asserted, the end_of_line and new_write_cmd signals are asserted, or the transaction has reached the last dword of data, the transaction involves a single dword. In this situation, the transaction terminates and the state machine returns to the IDLE state 2700 only when the target took the last piece of data (!p2q_trdy). Otherwise, the state machine remains in the WDATA2 state 2710. If the transaction involves more than one dword of data, the master state machine enters a WDATA2 burst data phase state 2710. Just before entering the WDATA2 state, the master state machine inserts a q2p_irdy wait state if the overflow_next_queue signal has been asserted.

In the WDATA2 state 2710, the master state machine bursts data to the PCI bus. If the p2q_ack signal is deasserted or the transaction is aborted by the PCI interface, the transaction is terminated in the QPIF and the master state machine reenters the IDLE state 2710. If the transaction is retried by the PCI interface but the PCI interface took the data provided (!p2_trdy), the master state machine reenters the IDLE state 2700. Otherwise, the state machine enters a WRETRY stepback state 2712 that steps the PMWQ out pointer back to the previous piece of data by generating the stepback signal discussed above. From the WRETRY state 2712, the state machine always reenters the IDLE state 2700.

If the p2q_ack signal remains asserted and the transaction is neither retried nor aborted, the master state machine determines whether the transaction is complete. If the QPIF indicates the end of the transaction (queue_cyc_complete is asserted) or the end of a cache line is reached and a new write command is needed (end_of_line and new_write_command are asserted), the state machine enters a WSHORT_BURST state 2714 when either the last piece of data was taken (!p2_trdy) or the PCI start pulse is received. In either case, only two dwords of data must be written to the PCI bus. Otherwise, the state machine remains in the WDATA2 state 2710. When the state machine enters the WSHORT_BURST state 2714, the QPIF frame_ signal remains asserted if the transaction can overflow into the next queue and a new write command is not needed.

In the WSHORT_BURST state 2714, the master state machine prepares to write the final two dwords of data to the PCI bus. If the p2q_ack signal is deasserted or the cycle is retried or aborted by the PCI interface, the state machine terminates the transaction and returns to the IDLE state 2700. When the PCI acknowledge signal disappears or the cycle is aborted, the master state machine asserts the stepback signal to indicate that the PMWQ out pointer should be stepped back to the previous dword. When the transaction is retried by the PCI interface, the out pointer is stepped

(continued)

Slave state transitions		
SLAVE STATE MACHINE		
CURRENT STATE	EVENT	NEXT STATE
	O: p2q_qcyc&&(dcq_no_data&& !p2q_irdy ‖ (pmw_counter[2] &&pmw_counter[1] && pmw_counter[0] && read_disconnect_for_stream) P: otherwise	HIT_DCQ_FINAL HIT_DCQ
HIT_DCQ_FINAL	Q: !p2q_qcyc ‖ !p2q_irdy R: otherwise	SLAVE_IDLE HIT_DCQ_FINAL
PMW1	S: !p2q_qcyc T: otherwise	SLAVE_IDLE PMW1

At reset, the slave state machine enters an IDLE state 2720, in which the QPIF waits for a transaction be initiated by a device on the PCI bus. If a transaction initiated on the bus does not target the QPIF (q2p_qcyc is not asserted), the slave state machine continues in the IDLE state 2720. When a transaction on the PCI bus does target the QPIF, the slave state machine enters a SLAVE_DAC dual address cycle state 2722 if the p2q_dac_flag is asserted and an address parity error has not occurred (p2q_perr_ is low). If the transaction is not a dual address cycle and is a posted memory write request, and if a parity error has not occurred in the address phase, the slave state machine loads the write counters (i.e., asserts load_write_counter) and determines whether it can accept the transaction. If the PMWQ in the other bridge chip is full (tc_dc_full is asserted by the DC transaction counter) or the DCQ is locked (dcq_locked is asserted) or the QPIF lock logic is in the unlocked-but-retry state (lock_state[1] equals "1"), the slave state machine terminates the transaction by asserting an asynchronous retry signal (early_retry) that is passed to PCI interface as q2pif_retry and remains in the IDLE state 2720. If the QPIF can accept the transaction, the slave state machine initiates the posted memory write message on the cable and enters a PMW1 state 2724, in which the transaction is forwarded up the cable.

If the transaction is not a dual address cycle or a posted memory write request, the slave state machine loads the dword counter (asserts load_write_counter) and, if no parity error has occurred, analyzes the delayed request transaction. If the transaction is a MRL or a MRM transaction and the QPIF lock logic is not in the unlocked-but-retry state, the slave state machine asserts the QPIF check cycle signal (q2pif_check_cyc), which instructs the DCQ to compare the latched request to the delayed completion messages in the DCQ buffers. If the request hits a DCQ buffer that is not empty (dcq_hit and !dcq_no_data), the slave state machine enters a STEP_AHEAD state 2726 in which the QPIF begins delivering the requested information to the PCI bus. If the MRL or MRM request misses all of the DCQ data buffers (!dcq_hit), the DCQ is not full (!dcq_full), the delayed request queue in the other bridge chip is not full (!tc_dr_full), and the DCQ and QPIF are not locked (!dcq_locked and !lock_state[1]), the slave state machine asserts the q2pif_retry signal, forwards the request down the cable, and remains in the IDLE state 2720. If the request misses the DCQ and the requost cannot the sent down the cable, the QPIF simply retries the requesting device and remains in the IDLE state 2720.

If the delayed request is not a MRL or MRM transaction, a second clock cycle is needed to check the request because the data or byte enables must be compared to the contents of the DCQ buffers, so the slave state machine enters a SECOND_CHECK state 2728. If a parity error occurs or if the lock logic is in the unlocked-but-retry state, the state machine retries the requesting device and remains in the IDLE state 2720.

In the SLAVE_DAC state 2722, the slave state machine receives the second half of the address phase information. If the requesting device has not targeted the QPIF, the slave state machine ignores the transaction and remains in the IDLE state 2720. When the QPIF is the target device, the state transition events are the same as those in the IDLE state 2720. Specifically, if the transaction is a posted memory write request and a parity error has not occurred, the slave state machine loads the write counters and determines whether it can accept the transaction. If the PMWQ in the other bridge chip is full (tc_pmw_full is asserted), the DCQ is locked, or the QPIF lock logic is in the unlocked-but-retry state, the slave state machine retries the requesting device and returns to the IDLE state 2720. If the QPIF can accept the transaction, the slave state machine initiates the posted memory write message on the cable and enters the PMW1 state 2724.

If the transaction is not a posted memory write request, the slave state machine loads the dword counter and, if no parity error has occurred, analyzes the delayed request transaction. If the transaction is a MRL or a MRM transaction

Cable Message Generator State Transitions		
CABLE MESSAGE GENERATOR		
CURRENT STATE	**EVENT**	**NEXT STATE**
CABLE_IDLE	A: (send_message && q2pif_dac) || ((dcq_prefetch_mul || dcq_prefetch_line) && dcq_prefetch_dac)	CABLE_DAC
	B: (send_message && !q2pif_dac) || ((dcq_prefetch_mul || dcq_prefetch_line) && !dcq_prefetch_dac) || (dcq_stream_connect && !(ldrq_valid[3:0])) && (dcq_stream_connect || !p2q_ack || dcq_prefetch_line || dcq_prefetch_mul)	SLAVE_DPHASE
	C: (send_message && !q2pif_dac) || ((dcq_prefetch_mul || dcq_prefetch_line) && !dcq_prefetch_dac) || ((dcq_stream_connect && !(ldrq_valid[3:0]))&& !dcq_stream_connect && !(!p2q_ack || dcq_prefetch_mul || dcq_prefetch_line)	MASTER_DPHASE
	D: otherwise	CABLE_IDLE
CABLE_DAC	E: !p2q_ack || dcq_prefetch_mul || dcq_prefetch_line	SLAVE_DPHASE
	F: otherwise	MASTER_DPHASE
MASTER_DPHASE	G: send_message &&q2pif_dac	CABLE_DAC
	H: send_message && !q2pif_dac	SLAVE_DPHASE
	I: !send_message &&(early_last_master_data && !p2q_trdy || master_abort_cable)	CABLE_IDLE
	J: otherwise	MASTER_DPHASE
SLAVE_DPHASE	K: [!(drq_stream_connect &&!(ldrq_valid[3:0]&&p2q_qcyc)] && [(dly_read_request || dly_single_write_request || dcq_prefetch_mul || dcq_prefetch_line)] ||	CABLE_IDLE
	L: early_last_slave_data || dcq_stream_connect &&!(ldrq_valid[3.0]) &&p2q_qcyc and otherwise	SLAVE_DPHASE

At reset, the cable message generator enters the IDLE state 2740, in which it waits for transaction information to arrive from the master or slave state machines. From the IDLE state 2740, if the cable message generator receives a prefetch multiple signal (dcq_prefetch_mul) or a prefetch line signal (dcq_prefetch_line), the cable address signal (early_cad[31:2]) equals the prefetch address signal (dcq_prefetch_addr[31:2]). Otherwise the early_cad[31:2] signal takes on the value of the QPIF address signal (q2pif_addr[31:2]). When the cable message is initiated by the master state machine, the message is a delayed completion message, so the command code (early_ccbe[3:0]) equals "1001". When the cable message is initiated by the slave state machine, the command code takes on the value of the message_cmd[3:0] signal, discussed above.

If the send_message signal is asserted, indicating that either the master state machine or the slave state machine has initiated a message, and the corresponding transaction is not a dual address cycle, or if the cable message generator receives a prefetch request that is not a dual address cycle, or if the cable message generator receives a stream connect signal and no delayed requests from the CPU are pending in the downstream DRQ, the cable message generator asserts a sent_pmw signal that indicates that a posted memory write request from the PCI bus will be sent down the cable. The sent_pmw signal is not asserted if a stream has been established by the DCQ. The cable message generator asserts a sent_dr signal when a read request or delayed write request is received from the slave state machine or a prefetch signal is received and when a stream has not been established by the DCQ.

If the DCQ has established a stream (dcq_stream_connect is asserted), the buffer number for the cable signal (early_cbuff[2:0]) takes on the value of the DCQ stream buffer (dcq_stream_buff[2:0]), the cable command code (early_ccbe[3:0]) is set equal to "1000", and the cable message generator enters the SLAVE_DPHASE state 2746. Otherwise, if the QPIF is in the slave mode and the cable message generator receives either a prefetch multiple or a prefetch line signal, the cable buffer signal takes on the value of the DCQ buffer number (dcq_buff[2:0]) and the cable message generator enters the SLAVE_DPHASE state 2746. Otherwise, the QPIF is operating in the master mode and the cable message generator enters the MASTER_DPHASE state 2744.

When the cable message generator receives the send_message signal and a transaction that is a dual address cycle, or when it receives a prefetch request that is a dual address cycle, the message generator enters the

PLL 180 locks to an incoming clock PCICLK2 from a clock buffer 181.

In the ensuing description, a "1X clock" refers to a clock having the same frequency as the clock PCICLK1, while a "3X clock" refers to a clock having three times the frequency of the clock PCICLK1. A 1X clock PCLK generated by the PLL 184 or 180 (in the bridge chip 26 or 48, respectively) is used for the corresponding bridge chip's PCI bus interface logic 188 or 190, and the 3X clock PCLK3 is used to run the cable message generation logic in the master cable interface 192 or 194. The other PLL 186 or 182 is used to lock to a cable input clock CABLE_CLK1 (from upstream) or CABLE_CLK2 (from downstream) and to generate a 1X clock CCLK and a 3X clock CCLK3 to capture incoming cable data. The clock outputs of the PLL 186 or 182 are routed to the slave cable interface 196 or 198, respectively.

The PLLs are arranged in the layout to balance the 1X and 3X clocks as closely as possible to minimize the skew between them.

The PLL 184 or 180 generates a phase indicator signal PCLKPHI1, which indicates to the master cable interface 192 or 194 when the first phase of data should be presented to the cable 28. On the upstream side, the signal PCLKPHI1 is based on the PCI clock PCICLK1; on the downstream side, the signal PCLKPHI1 is based on the PCI clock PCICLK2. The PLL 186 or 182 generates a phase indicator signal CCLKPHI1, based on the cable clock CABLE_CLK1 or CABLE_CLK2, to indicate to the slave cable interface 196 or 198 when the first phase of data has come down the cable 28.

The PCI clock PCICLK2 for the secondary PCI bus 32 is generated off a 1X clock BUFCLK of the PLL 182 in the downstream bridge chip 48. The clock BUFCLK drives the clock buffer 181 through a driver 179. The buffer 181 outputs a separate clock signal for each of the six slots on the secondary PCI bus 32 as well as the clock PCICLK2, which is routed back as the bus input clock to the downstream bridge chip 48. By basing the clock PCLK on the clock PCICLK2 from the clock buffer 181, the clock schemes of the upstream and downstream chips are made to appear more similar since both are based on an external bus clock.

The cable clock CABLE_CLK1 is a 33% duty cycle clock. The PLL 182 first converts the 33% duty cycle clock to a 50% duty cycle clock for output as BUFCLK.

The PCI Specification, Version 2.1, requires that the PCI bus clock must meet the following requirements: clock cycle time greater than or equal to 30 ns; clock high time greater than 11 ns; clock low time greater than or equal to 11 ns; and clock slew rate between 1 and 4 ns.

When the computer system is powered up, the upstream chip 26 is powered on last, the upstream PLL 184 sends the clock CABLE_CLK1 (through the master interface 192) down the cable 28, which is then locked to by the downstream PLL 182 and PLL 180. The downstream PLL 180 then sends the clock CABLE_CLK2 back upstream to be locked to by the PLL 186. The system is not completely operational until all four PLLs have acquired lock.

If the upstream bridge chip 26 powers up and the downstream bridge chip 48 is not yet turned on, the upstream bridge chip 26 behaves as a PCI-PCI bridge with nothing connected to its downstream bus (the cable 28). As a result, the upstream bridge chip 26 does not accept any cycles until the downstream bridge chip 48 is powered on and the upstream PLL 186 has acquired "lock" from the cable clock CABLE_CLK2.

The upstream bridge chip 26 floats all of its PCI output buffers and state machines asynchronously with assertion of the PCI reset signal PCIRST1_ on the primary bus 24. During reset, the PLL 184 may be attempting to acquire lock on the PCI bus clock PCICLK1. Since the PCI Specification guarantees that the signal PCIRST1_ will remain active for at least 100 μs after the PCI bus clock becomes stable, the PLL 184 has about 100 μs to acquire a lock.

The downstream bridge chip 48 resets all internal state machines upon detection of the primary bus PCIRST1_ signal. In response, the downstream bridge chip 48 also asserts a slot-specific reset to each slot on the secondary PCI bus 32, as well as a secondary PCI bus reset signal PCIRST2_.

Referring to Figure 6, each PLL includes a voltage-controlled oscillator (VCO) 200 generating an output 201 (the 3X clock) between 75 Mhz (for a 25-Mhz PCI bus) and 100 Mhz (for a 33-Mhz PCI bus). The VCO 200 receives a reference clock 197, which is the PCI bus clock. Each PLL has a lock detection circuit 205 which indicates by a lock indication bit that the PLL phase is locked to its reference accurately enough to perform its intended function.

The lock indication bits are written to a status register in the configuration space 105 or 125 of each bridge chip. On the downstream side, a power-good/lock status bit is transmitted to the upstream bridge chip 26 to indicate that the main elements of the downstream bridge chip 48 are stable (power is stable) and the downstream PLLs are locked (lock indication bits of the two PLLs are active). The lock indication bit is also gated with the EDC status bits such that EDC errors are not reported as such until the PLLs are locked. Thus, the bridge chip pair can come up to an error-free communication state without software intervention. The lock indication bit also provides some diagnostic information which can distinguish between a PLL lock failure and other data errors. The clock generation circuitry includes a four-state machine 202 to generate a divide-by-3 clock (1X clock) of the VCO output 201. The 1X clock is fed back to the PLL at input 203.

Data is moved down the cable 28 at a 3X clock (PCLK3) rate in three time-multiplexed phases to produce a 1X clock message transfer rate. Referring to Figure 7, the circuitry in the master cable interface 192 or 194 for disassembling and transmitting the cable message includes a register 204, which samples the out-going message at the local

from two different PLLs. On the downstream side, the signal EN_OUTCNT is the signal EN_INCNT latched on the negative edge of the signal PCLK by the flip-flop 230. A multiplexer 234 selects the output of the flip-flop 230 since the signal UPSTREAM_CHIP is low.

In the upstream bridge chip 26, the cable interface is treated as completely asynchronous. The phase uncertainty is due to the unknown phase shift of the cable 28 itself. Designing for this uncertainty gives complete freedom on the length of cable 28. What is known is that the clocks in the upstream and downstream bridge chips have the same frequency, since they both have their origin in the upstream PCI bus clock PCICLK1. In the upstream bridge chip 26, the signal EN_OUTCNT is the signal EN_INCNT latched on the positive edge of the clock PCLK by a flip-flop 232. The multiplexer 234 selects the output of the flip-flop 232 since the signal UPSTREAM_CHIP is high. The flip-flop 232 guarantees that even for the worst-case lineup of the cable clock CABLE_CLK2 and the local PCI clock PCLK (one complete PCLK period phase shift), there is valid data in the FIFO 216 before the data is transmitted to the rest of the chip.

Referring to Figure 11, the cable data is received by the slave cable interface 196 or 198 as three phase time-multiplexed signals A1, A2 and A3; B1, B2 and B3; C1, C2 and C3; and so forth. A previous transaction is completed in periods T0, T1 and T2. Beginning in period T3, the first phase data A1 is presented to the register 218 and the first phase indicator CCLKPHI1 is pulsed high. On the falling edge of CCLK3 in period T3, the data A1 is loaded into the register 218, and the local phase 1 indication signal PHI_DLY is pulsed high. In period T4, on the falling edge of clock, the phase 1 data A1 is loaded into the register 220, the phase 2 data A2 is loaded into the register 218, and the phase 2 indication signal PHI2_DLY is pulsed high. In period T5, on the falling edge of CCLK3, the phase 2 data is loaded into the register 222, the phase 3 data A3 is loaded into the register 218, and the phase 3 indication signal PHI3_DLY is pulsed high. In period T6, the contents of the registers 220, 222, and 218 are loaded into the selected entry of the FIFO 216 on the following edge of CCLK3. Also in period T6, the data B1 is presented to the register 218 along with the indication signal CCLKPHI1. Messages B and C are loaded into the FIFO 216 in the same manner as message A in subsequent periods.

Referring to Figure 12, the input pointer INPTR[1:0] starts at the value 0 in period T0 on the rising edge of the clock CCLK3. Also in period T0, message A is loaded into FIFO 0 on the falling edge of the clock CCLK3. In the downstream bridge chip 48, the output pointer OUTPTR[1:0] is incremented to the value 0 on the next rising edge of the clock PCLK in period T3. Also in period T3, the input pointer INPTR[1:0] is incremented to the value 1 on the rising edge of the clock CCLK3, and message B is loaded into FIFO 1 on the falling edge of CCLK3. Cable data is thus loaded into FIFO0, FIFO1, FIFO2, and FIFO3 in a circular fashion.

On the upstream side, if the input pointer INPTR[1:0] is 0 in period t0, the output pointer OUTPTR[1:0] is incremented to the value 0 in period T6, two PCLK periods after the input pointer INPTR[1:0]. The two PCLK period lag in the upstream bridge chip 26 allows the phase delay in the cable 28 to be of any value, which has the advantage that the cable length need not be of a specific fixed value.

Referring to Figure 13, the input and output flip flops on the cable interface are custom-placed by the manufacturer of the chips to minimize the skew between the cable data and the clock passed with it. The amount of wire between each flip-flop and the I/O are maintained as consistent as possible between all cable interface signals.

CABLE MESSAGE

Sixty bits of cable data constitute one message. The 60 bits are multiplexed onto 20 cable lines and are transmitted each 10 ns over the cable 28. The table in Figure 14 shows the bits and the phase each bit is assigned to. The first three columns show the upstream-to-downstream data transfer format, and the last three columns show the downstream-to-upstream data transfer format. The following is a description of the signals.

EDC[7:0]: The signals are the eight syndrome bits used to detect and correct errors encountered in transmitting data over the cable 28.

CAD[31:0]: The signals are the 32 address or data bits.

CFRAME_: The signal is used to signal the start and end of a cable transaction, similar to the PCI FRAME_signal.

CCBE[3:0]_: The four bits form byte enables in some PCI clock phases and either a PCI command or a message code in other PCI clock phases.

CBUFF[3:0]: In the address phase, the signals indicate a buffer number for initializing the bridge chip delayed completion queue (DCQ) 148 to tie upstream and downstream delayed read completion (DRC) and delayed read request (DRR) transactions. After the address phase, the signals contain the parity bit, parity error indication and the data ready signal.

COMPLETION REMOVED: The bit is used to signal that a delayed completion has been removed from the transaction ordering queue (TOQ) on the other side of the cable 28.

PMW ACKNOWLEDGE: The bit is used to signal that a posted memory write (PMW) has been completed on the other side and has been removed from the transaction run queue (TRQ).

of the first are standardization, ruggedness and reliable uniform manufacture. The advantages of the second are greater mechanical flexibility, automatic termination to the connector in assembly and possibly lower cost.

The table of Figure 16 shows some of the HIPPI cable specifications. The ground shield consists of a braid over aluminum tape and carries only minimum DC currents due to the differential nature of the buffers to be used. The method of signaling is true differential which provides several advantages, with differential buffers used to send and receive signals over the cable 28. First the true differential method is less expensive than fiber optics for this short distance and less complex to interface than other serial methods. Differential signaling provides significant common mode noise immunity and common mode operating range, is available in ASICs and is faster than TTL. When using twisted pair and shielding, it minimizes electromagnetic radiation. When using low voltage swings, it minimizes power dissipation.

The signaling levels chosen as a target are described in the IEEE Draft Standard for Low-Voltage Differential Signals (LVDS) for Scaleable Coherent Interface (SCI), Draft 1.10 (May 5, 1995).

The cable connector is an AMP metallic shell 100-pin connector with two rows of pins. The rows are 100 mils apart and the pins are on 50-mil centers. The metal shell provides EMI shielding and the connection of the ground path from the cable shield to the board connector. The mating right angle board connector just fits a PCI bracket. The connector is to have a bar running between the rows of pins to divert electrostatic discharges from the signal pins when the connector is disconnected. A pair of thumb screws attached to the cable connector will secure the mated connectors.

ERROR DETECTION AND CORRECTION

An error detection and correction (EDC) method is implemented on each bridge chip to protect communication over the cable 28. Since the data is time-multiplexed into three 20-bit groups to be sent over 20 pairs of wires, each triplet of "adjacent" bits (i.e., bits associated with the same wire in the cable 28) is arranged so as to be transmitted on a single wire pair. The EDC method can correct single-bit failures and multi-bit failures occurring in the same bit position in each of the three time-multiplexed phases. The multi-bit failures are typically associated with a hardware failure, e.g., a broken or defective wire or a faulty pin on bridge chips 26, 48.

Twenty wire pairs of the cable 28 are used for downstream communication and 20 more for upstream communication. For the remaining ten pairs in the 50-pair HIPPI cable 28 (which pass such information as the clock signals CABLE_CLK1 and CABLE_CLK2, reset signals, and the power good/PLL-lock signal), error detection and correction is not implemented.

The following are the underlying assumptions for the EDC algorithm. Most errors are single bit errors. The probability of having random multiple-bit errors in the same transaction is extremely remote because the cable 28 is not susceptible to interference from internal or external sources. Errors caused by a defective wire may affect a single bit or a group of bits transmitted on that wire. When a hardware failure occurs, the logic state of the corresponding differential buffer is in a single valid logic state.

Referring to Figure 17, the output signals FIFOOUT[59:0] from the multiplexer 228 in the slave cable interface 196 or 198 are provided to the input of a check bit generator 350, which produces check bits CHKBIT[7:0]. The check bits are generated according to the parity-check matrix shown in Figure 18, in which the first row corresponds to CHKBIT [0], the second row corresponds to CHKBIT[1], and so forth. The bits across a row correspond to data bits FIFOOUT [0:59].

The check bits are generated by an exclusive-OR of all the data bits FIFOOUT[X] (X is equal to 0-59), which have a "1" value in the parity-check matrix. Thus, the check bit CHKBIT[0] is an exclusive-OR of data bits FIFOOUT[7], FIFOOUT[8], FIFOOUT[9], FIFOOUT[12], FIFOOUT[13], FIFOUT[16], FIFOOUT[22], FIFOOUT[23], FIFOOUT[24], FIFOOUT[26], FIFOOUT[32], FIFOOUT[33], FIFOOUT[34], FIFOOUT[35], FIFOOUT[38], FIFOOUT[39], FIFOOUT [45], FIFOOUT[46], FIFOOUT[48], FIFOOUT[49], FIFOOUT[51], and FIFOOUT[52]. Similarly, the check bit CHKBIT [1] is an exclusive-OR of bits 0, 1, 4, 5, 9, 10, 12, 14, 15, 16, 23, 27, 35, 37, 38, 40, 43, 46, 47, 48, 50, and 53. Check bits CHKBIT[2:7] are generated in similar fashion according to the parity-check matrix of Figure 18. The parity check matrix is based upon the 20 sub-channels or wires per time-multiplexed phase and a probability that multiple errors in the accumulated data are attributable to a faulty subchannel or wire that affects the same data position in each time-multiplexed phase.

In the master cable interface 192 or 194, the check bits CHKBIT[7:0] are provided as error detection and correction bits EDC[7:0] along with other cable data to allow error correction logic in the slave cable interface 196 or 198 to detect and correct data errors.

The check bits CHKBIT[7:0] are provided to a fix bit generator 352, which generates fix bits FIXBIT[59:0] according to the syndrome table shown in Figure 19. The check bits CHKBIT[7:0] have 256 (2^8) possible values. The syndrome table in Figure 19 contains 256 possible positions. Each of the 256 positions in the syndrome table contains 2 entries, the first entry being the hexadecimal value of the check bits CHKBIT[7:0], and the second entry indicating the cable data status associated with that position. Thus, for example, a hexadecimal value 00 indicates a no-error condition, a

the PCI arbiter 124. If the PCI arbiter 124 is disabled, then an external arbiter (not shown) is used and the hot plug request is driven out on the REQ[1]_ pin and hot plug grant is input on the GNT[1]_ pin. The bridge PCI bus request is driven out on the REQ[2]_ pin and its grant is input on the GNT[2]_ pin. If the bridge chip 48 samples REQ[7]_ high on PCIRST2_, it will enable the PCI arbiter 124.

The PCI arbiter 124 negates a master's GNT_ signal either to service a higher priority initiator, or in response to the master's REQ_ signal being negated. Once its GNT_ signal is negated, the current bus master maintains ownership of the bus until the bus returns to idle.

If no PCI agents are currently using or requesting the bus, the PCI arbiter 124 does one of two things depending on the value of a PARKMSTRSEL configuration register in the configuration space 125. If the register contains the value 0, the PCI arbiter 124 uses the last active master to park on the bus 32; if it contains the value 1, then the bus is parked at the bridge chip 48.

The PCI arbiter 124 includes a PCI minimum grant timer 304 (Figure 21) which controls the minimum active time of all the GNT_ signals. The default value for the timer 304 is the hexadecimal value 0000 which indicates that there is no minimum grant time requirement. The timer 304 can be programmed with a value from 1 to 255, to indicate the number of PCICLK2 clock periods the GNT_ line is active. Alternatively, an individual minimum grant timer can be assigned to each PCI master on the secondary bus 32 to provide more flexibility. The minimum grant time is applicable only when the current master is asserting its REQ_ signal. Once the REQ_ signal is deasserted, the GNT_ signal can be removed regardless of the minimum grant time value.

Referring to Figure 20A, in normal operation, the PCI arbiter 124 implements a round-robin priority scheme (second level arbitration scheme). The eight masters in the round-robin scheme include devices connected to the six slots of the expansion box 30, the SIO 50, and a posted memory write (PMW) request from the upstream bridge chip 26. All masters on the PCI bus 32 in this scheme have the same priority as the bridge chip 48. After a master has been granted the secondary PCI bus 32 and the master has asserted the FRAME_ signal, the bus is re-arbitrated and the current master is put on the bottom of the round-robin stack. If the master negates its request or the minimum grant timer 304 expires, the PCI bus 32 is granted to the next highest priority master. Locked cycles are not treated any differently by the PCI arbiter 124.

In response to certain events, the arbitration scheme is modified to optimize system performance. The events include: 1) an upstream-to-downstream delayed read or delayed write request is pending, 2) a downstream-to-upstream delayed read request is pending with no read completion indication provided, and 3) a streaming possibility exists while the bridge chip 26 is the current master on the upstream bus 24.

When a delayed request is detected, the bridge chip 48 becomes the next master to be granted the secondary PCI bus 32. Once the bridge chip 48 is granted the bus 32, it maintains ownership of the bus 32 until it completes all outstanding delayed requests or one of its cycles is retried. If the bridge chip 48 is retried, then a two-level arbitration scheme is implemented by the arbiter 124. One primary cause of the bridge chip read cycle being retried is that the target device is a bridge with a posted write buffer that needs to be flushed. In this case, the optimum operation is to grant the bus 32 to the retrying target to allow it to empty its posted write buffer so it can accept the bridge chip read request.

Referring to Figure 20B, the two-level arbitration protocol includes a first level arbitration scheme which is a round-robin scheme among three possible masters: the delayed request from the CPU 14, a request from the retrying master, and a master selected by the second-level arbitration scheme. Each of the three masters in the first-level arbitration scheme is guaranteed every third arbitration slot. For memory cycles, the slot associated with the retrying target can be determined from target memory range configuration registers in the configuration space 125 of the bridge chip 48, which store the memory range associated with each PCI device. If the retrying master cannot be determined (as in the case of an I/O read), or if the retrying master is not requesting the secondary bus 32, then the first level arbitration scheme would be between the bridge chip 48 and a level-two master.

The retrying master is not masked from the level-two arbitration. Thus, it is possible for it to have two back-to-back arbitration wins if it is the next master in the level-two arbitration scheme.

For example, if an upstream-to-downstream read is retried and Master C (the retrying master) is requesting the bus 32 as well as Master B and Master E, the order of the bus grants would be as follows in descending order: the bridge chip 48, the retrying master (Master C), Master C, the bridge chip 48, the retrying master C, Master E, the bridge chip 48, and so forth, until the bridge chip 48 is able to complete its transaction and the PCI arbiter 124 reverts back to its level-two arbitration scheme for normal operation.

If, as another example, the bridge chip read is retried and the only other requesting masters are Master A and Master D (i.e., the retrying master is not requesting the bus or it could not be identified because it is accessing I/O space), the order of the bus grants is as follows: the bridge chip 48, Master A, the bridge chip 48, Master D, and so forth.

The two-level arbitration scheme gives delayed requests from the CPU 14 the highest priority. Although this arbitration method favors heavily the CPU 14, every requesting device on the bus 32 is eventually granted the PCI bus 32. By so doing, there is less chance that the other secondary bus masters would be starved when a PCI bridge chip

From state IDLE4GNT, the grant state machine 306 next transitions to the GNT state, and the signals GNT[7:0] are set to the state of new grant signals NEWGNT[7:0] and the signal CHANGING_GNT is negated low. The signals NEWGNT[7:0] are based on the state of the current master signals CURMAST[2:0], as shown in Figure 24.

From state GNT, three transitions are possible. The grant state machine 306 returns to the PARK state if an arbitration window is open (OPEN_WINDOW is high), no request is pending (ANY_REQ is low), the PCI bus 32 is idle (BUS_IDLE is high), and the next master is the current master (i.e., the current master is the parking master). In the transition back from the GNT state to the PARK state, the signals LISTATE[1:0] are updated with the signals N_L1STATE [1:0]. However, if no requests are pending and the bus is idle, but the current master is not the parking master (i.e., the signals N_CURMAST[2:0] are not equal to the value of the signals CURMAST[2:0]), an idle state is needed and the grant state machine 306 transitions from the GNT state to the IDLE4PARK state. The L1 state values L1STATE[1:0] are updated. From the IDLE4PARK state, the grant state machine 306 transitions to the PARK state, setting the grant signals GNT[7:0] equal to the new grant signals NEWGNT[7:0] to grant the PCI bus 32 to the new master. The signal CHANGING_GNT is also negated low.

If the arbitration window opens up (OPEN_WINDOW is high), and the next master is not the current master (the signals N_CURMAST[2:0] are not equal to the signals CURMAST[2:0]), then the grant state machine 306 transitions to the idle state IDLE4GNT to change bus master grants. In the transition, the signal CHANGING_GNT is asserted high, the signals GNT[7:0] are cleared to all zeros, the signals CURMAST[2:0] are updated with the next master value N_CURMAST[2:0], and the L1 state signals L1STATE[1:0] are updated with the next state value N_L1STATE[1:0]. In addition, the round-robin master signals RR_MAST[2:0] are updated with the next round-robin master N_RR_MAST [2:0] if the signal ADV_RR_MAST is asserted high. The grant signals GNT[7:0] are then assigned to the value NEWGNT [7:0] in the transition from the IDLE4GNT state to the GNT state.

Referring to Figure 23, the L1 state machine 300 (Figure 21) starts in state RR upon assertion of the RESET signal, where the state machine 300 remains while a delayed request signal BAL_DEL_REQ is negated low (indicating there is no delayed request pending). While in the RR state, the signal ADV_RR_MAST is asserted high to allow the grant state machine 306 to update the round-robin master (i.e., setting signals RR_MAST[2:0] equal to the value N_RR_MAST[2:0]. The RR state is the round-robin state in which the level-two arbitration scheme is used. While in the RR state, the next master signals N_CURMAST[2:0] are set equal to the next round-robin master N_RR_MAST [2:0], and the signal OPEN_WINDOW is set high if a stream request opportunity exists (STREAM_REQ is high), or the minimum grant timer 304 has expired (MIN_GRANT is high), or the current master has negated its request (CURMAST_REQ goes low). When asserted high, the signal OPEN_WINDOW allows a new arbitration to take place.

If a delayed request is detected (BAL_DEL _REQ goes high), the L1 state machine 300 transitions from the RR state to the BAL state, setting the next master state N_CURMAST[2:0] as the bridge chip 48 and deasserting the signal ADV_RR_MAST to disable the level-two round-robin arbitration. In the BAL state, the signal OPEN_WINDOW is asserted high if the delayed request is deasserted (BAL_DEL_REQ goes low) or the delayed request has been retried (BAL_RETRIED goes high). If the signal BAL_DEL_REQ is negated low, or if the delayed request BAL_DEL_REQ is asserted high but the retrying master request is negated low (RTRYMAST_REQ is low) and the slot request ANY_SLOT_REQ is asserted high, then the L1 state machine 300 transitions back to the RR state. In the transition, the signal ADV_RR_MAST is asserted high and the next master signals N_CURMAST[2:0] are set equal to the next round-robin master N_RR_MAST[2:0]. If the signal BAL_DEL_REQ is deasserted, that indicates that the arbiter 124 should revert back to the level-two round-robin scheme. If the delayed request signal is asserted but the retrying master request is negated, then the level-one arbitration scheme is between the slots on the PCI bus 32 and the bridge chip 48.

If both the delayed request BAL_DEL_REQ and the retrying master request RTRYMAST_REQ are asserted, then the L1 state machine 300 transitions from state BAL to state RETRY_MAST, and the retrying master is set as the next master (N_CURMAST[2:0] is set equal to RTRY_MAST[2:0]). The signal ADV_RR_MAST is maintained low. In the RETRY_MAST state, if none of the PCI slot masters are asserting a request (ANY_SLOT_REQ is low), then the level-one arbitration scheme is between the retrying master and the bridge chip 48, and the L1 state machine 300 transitions back to the BAL state. The bridge chip 48 is set as the next master (N_CURMAST[2:0] is equal to the state BALBOA), and the signal ADV_RR_MAST is maintained low. However, the L1 state machine 300 transitions from the RETRY_MAST state to the RR state if any one of the slot masters is asserting a request (ANY_SLOT_REQ is high). In the transition, the signal ADV_RR_MAST is asserted high, and the next round-robin master is set as the next master (N_CURMAST[2:0] is set equal N_RR_MAST[2:0]).

To take advantage of the streaming capabilities of the bridge chip, when data for a DRC starts arriving from the cable 28, the master associated with that DRC becomes the highest priority device (assuming its REQ_ is asserted). This allows the master to receive the data stream coming down the cable 28 while the window of opportunity is there for streaming. If the bridge chip 48 cannot connect the master before the DRC queue fills up, then the upstream bridge chip 24 will disconnect and only a portion of the data would be passed to the requesting master, necessitating the master to issue another read request on the upstream bus 24. The streaming master retains the highest priority as long as DRC data continues to arrive from the cable 28. If the master repeats a different cycle/address, it will be retried,

corresponding one of the bits MULTI_MASTER[7:1] is set high if the signal DCQ_HIT is low and the signal Q2PIF_CHECK_CYC is high. Thus, for example, if the signal Q2PIF_SLOT[2:0] contains the value 2, indicating that the device in slot 2 is the current master of the delayed request, and DCQ buffer 5 is storing a pending request for the slot 2 master (Q5_MASTER[2:0] = 5), and either of signals Q5_COMPLETE or Q5_PART_COMPLETE is high, and if the signal Q2PIF_CHECK_CYC is high and the signal DCQ_HIT is low, then the bit MULTI_MASTER[2] is set high to indicate that the slot 2 device is a multi-threaded master.

A mask request generation block 332 produces signals Q2A_MASK_REQ[X] (X = 1-7) in response to signals Q [7:0]_MASTER[2:0], Q[7:0]_STATE[3:0] (which indicates the state of delayed completion queues 0-7), SLOT_WITH_DATA[7:0] (which indicate if delayed completion Qs 0-7 contain valid data), CFG2Q_MULTI_MASTER [X] (X = 1-7), CFG2Q_ALWAYS_MASK, and CFG2Q_NEVER_MASK.

Referring to Figure 26A, the mask request generation block 332 includes a 2:1 multiplexer 320 for producing the signal Q2A_MASK_REQ[X] (X = 1-7). The 1 input of the multiplexer 320 is connected to the output of an OR gate 322 and the 0 input is tied low. The select input of the multiplexer 320 is driven by a signal MASK_MUXSEL. One input of the OR gate 322 is connected to the output of a NOR gate 324, which receives a signal CFG2Q_MULTI_MASTER[X] (indicating a multi-threaded master), and the other input receives a signal CFG2Q_NEVER_MASK (a configuration bit indicating that the request line should not be masked if a multi-threaded master is detected). The other input of the OR gate 322 receives a signal CFG2Q_ALWAYS_MASK, which is a configuration bit indicating that the corresponding mask bit Q2A_MASK_REQ[X] should always be masked if the signal MUXSEL is asserted high. The signal MASK_MUXSEL is asserted high if the request from the secondary bus master is not to data already existing in the queue block 127, i.e., the request must be transmitted to the primary PCI bus 24. Thus each time a request is transmitted from a device on the secondary PCI bus 32 upstream to the primary PCI bus 24, a check is performed on bits CFG2Q_MULTI_MASTER[7:1] to determine if a multi-threaded master has been detected.

The masking of requests can be overridden by setting the appropriate bits in the configuration registers 125. The available modes include: 1) normal mode in which request masking is enabled except if multi-threaded master (CFG2Q_NEVER_MASK = 0, CFG2Q_ALWAYS_MASK = 0), 2) always mask mode in which requests from retried masters are masked even if multi-threaded (CFG2Q_ALWAYS_MASK = 1), and 3) never mask mode in which the requests are never masked (CFG2Q_NEVER_MASK = 1, CFG2Q_ALWAYS_MASKED = 0).

EXPANSION CARD INSERTION AND REMOVAL CONNECTING EXPANSION CARDS

As shown in Figures 1 and 27A, the two expansion boxes 30a and 30b, of common design 30, each have the six hot-plug slots 36 (36a-f) in which the conventional expansion cards 807 can be inserted and removed (hot-plugged) while the computer system 10 remains powered up. The six mechanical levers 802 are used to selectively secure (when closed, or latched) the expansion cards 807 that are inserted into corresponding hot-plug slots 36. For purposes of removing or inserting the expansion card 807 into one of the slots 36, the corresponding lever 802 must be opened, or unlatched, and as long as the lever 802 is opened, the corresponding slot 36 remains powered down.

When the lever 802 that secures the expansion card 807 to its slot 36 is opened, the computer system 10 senses this occurrence and powers down the card 807 (and corresponding slot 36) before the card 807 can be removed from its slot 36. Slots 36 that are powered down, like other slots 36 not holding cards 807, remain powered down until software of the computer system 10 selectively powers up the slots 36.

The card 46 inserted into the card slot 34 has the bridge chip 48 that monitors the securement status (open or closed) of the levers 802 and powers down any card 807 (and corresponding slot 36) that is not secured by its lever 802. Software of the computer system 10 can also selectively power down any one of the slots 36.

The cards 807 are powered up through a power up sequence and powered down through a power down sequence. In the power up sequence, power is first supplied to the card 807 being powered up, and thereafter, a PCI clock signal (from the PCI bus 32) is furnished to the card 807 being powered up. Remaining PCI bus signal lines of the card 807 are then coupled to corresponding lines of the PCI bus 32. Lastly, the reset signal for the card 807 being powered up is negated which brings the card 807 out of reset.

The power up sequence allows the circuitry of the card 807 being powered up to become fully functional with the PCI clock signal before the remaining PCI bus signals are provided. When the clock signal and remaining PCI bus signals are connected to the card 807 and before the card 807 is reset, the bridge chip 48 has control of the PCI bus 32. Because the bridge chip 48 has control of the PCI bus 32 during these times, potential glitches on the PCI bus 32 from the power up sequence do not disturb operations of the 'cards 807 that are powered up.

In the power down sequence, the card 807 being powered down is first reset. Next, the PCI bus signals, excluding the PCI clock signal, are removed from the card 807. The bridge chip 48 subsequently disconnects the PCI clock signal from the card 807 before power from the card 807 is removed. The power down sequence minimizes the propagation of false signals from the card 807 being powered down to the bus 32 because circuitry on the card 807 remains fully functional until the PCI bus signal lines are removed.

(continued)

PARALLEL OUTPUT CONTROL SIGNALS (POUT [39:0])			
SIGNAL POSITION	DESCRIPTION	ASSOCIATED CONTROL SIGNALS	WHEN SIGNAL IS ACTIVE
36	Power enable signal for slot 36c	(PWREN [2])	High
37	Power enable signal for slot 36d	(PWREN [3])	High
38	Power enable signal for slot 36e	(PWREN [4])	High
39	Power enable signal for slot 36f	(PWREN [5])	High

As shown in Figures 2 and 28, each hot-plug slot 36 has the associated switch circuitry 41 for connecting and disconnecting the slot 36 to and from the PCI bus 32. The switch circuitry 41 for each slot 36 receives four of the control signals POUT[39:16]. As an example, for the slot 36a, when the control signal POUT[28] is asserted, or low, the slot 36a is connected to the bus signal lines of the PCI bus 32 by a switch circuit 47. When the control signal POUT[28] is deasserted, or high, the slot 36a is disconnected from the bus signal lines of the PCI bus 32.

When the control signal POUT[22] is asserted, or low, the slot 36a is connected to a PCI clock signal CLK through a switch circuit 43. When the control signal POUT[22] is deasserted, or high, the slot 36a is disconnected from the clock signal CLK.

When the control signal POUT[34] is asserted, or high, the slot 36a is connected to a card voltage supply level V_{SS} through a switch circuit 45. When the control signal POUT[34] is deasserted, or low, the slot 36a is disconnected from the card voltage supply level V_{SS}.

When the control signal POUT[16] is asserted, or low, the slot 36a is reset and when the control signal POUT[16] is deasserted, or high, the slot 36a comes out of the reset state.

As seen in Figure 2, the SIO circuit 50 may selectively monitor up to one hundred twenty-eight (sixteen bytes) of latched status signals STATUS[127:0] furnished by the expansion box 30. The status signals STATUS[127:0] form a "snapshot" of selected conditions of the expansion box 30. The status signals STATUS[127:0] include six status signals STATUS[5:0] which indicate the securement status (opened or closed) of each of the levers 802. The SIO circuit 50 monitors the status signals STATUS[31:0] for changes in their logical voltage levels. The SIO circuit 50 serially shifts the status signals STATUS[127:32] into the SIO circuit 50 when instructed to do so by the CPU 14.

The SIO circuit 50 serially receives the status signals STATUS[127:0], least significant signal first, via a serial data signal NEW_CSID. The data signal NEW_CSID is furnished by the serial output of the thirty-two bit, parallel input shift register 82 located on board the expansion box 30 along with the slots 36.

The register 82, through its parallel inputs, receives twenty-four parallel status signals PIN[23:0], four associated with each of the hot-plug slots 36, that are included in the thirty-two least significant status signals STATUS[31:0]. When the status indicated by one or more of the status signals STATUS[31:0] changes (the logical voltage level changes), the bridge chip 48 generates an interrupt request to the CPU 14 by asserting, or driving low, a serial interrupt request signal SI_INTR# which is received by the interrupt receiving block 132. The status signals PIN[23:0] include two PCI card presence signals (PRSNT1# and PRSNT2#) associated with each slot 36.

Six status signals PIN[5:0], corresponding to their latched versions, status signals STATUS[5:0], indicate the securement, or engagement, status (open or closed) of each the levers 802. Six sliding switches 805 (Figs. 27A-27C) are actuated by the movement of their corresponding levers 802 and are used to electrically indicate the securement status of the corresponding lever 802. Each switch 805 has a first terminal coupled to ground and a second terminal furnishing the corresponding one of the status signals PIN[5:0]. The second terminal is coupled to a supply voltage level V_{DD} through one of six resistors 801.

If one of the levers 802 opens and the card 807 secured by the lever 802 becomes unsecured, the corresponding one of the status signals PIN[5:0] is asserted, or driven high. As an example, for the slot 36a, the status signal PIN[0] is deasserted, or driven low, when the corresponding lever 802 is closed. When the lever 802 for the slot 36a is opened, the status signal PIN[0] is asserted, or driven high.

The register 82 also receives a serial stream of latched status signals STATUS[127:32] that do not cause interrupts when the logical voltage level of one of the signals STATUS[127:32] changes. The status signals STATUS[127:32] are formed by the sixteen bit shift register 52 located on board the expansion box 30 with the slots 36. The shift register

(continued)

STATUS [127:0]	
BIT	DESCRIPTION
32-127	Status signals that do not cause interrupt requests when their status changes

As shown in Figures 2 and 30, when the SIO circuit 50 asserts, or drives low, a register load signal CSIL_O_, the shift register 52 latches the status signals STATUS[127:32] and the shift register 82 latches the status signals STATUS [31:0]. When the SIO circuit 50 negates, or drives high, the signal CSIL_O_, both the registers 52 and 82 serially shift their data to the SIO circuit 50 on the positive edge of a clock signal CSIC_O furnished by the SIO circuit 50. The clock signal CSIC_O is synchronized to and one fourth the frequency of the PCI clock signal CLK.

As shown in Figure 29, for purposes of monitoring, or scanning, the status signals STATUS[31:0], the SIO circuit 50 uses a thirty-two bit interrupt register 800 whose bit positions correspond to the signals STATUS[31:0]. The SIO circuit 50 updates the bits of the interrupt register 800 to equal the corresponding status signals STATUS[31:0] that have been debounced, as further described below. Two status signals STATUS[7:6] are reserved for additional hot-plug slots 36, and the seventh and eighth most significant bits of the interrupt register 800 are also reserved for the additional slots 36. The interrupt register 800 is part of a register logic block 808 of the SIO circuit 50 which is coupled to the PCI bus 32.

Serial scan input logic 804 of the SIO circuit 50 sequentially scans, or monitors, the status signals STATUS[31:0], least significant signal first, for changes, as indicated by transitions in their logical voltage levels. If the status of one or more of the status signals STATUS[5:0] associated with the levers 802 changes, the serial scan input logic 804 enters a slow scan mode such that the status signals STATUS[5:0] are scanned thirty-two times within a predetermined debounce time interval. If one or more of the status signals STATUS[5:0] changes, the serial scan input logic 804 updates the interrupt register 800 (and asserts the serial interrupt signal SI_INTR#) if the changed status signal STATUS [5:0] remains at the same logical voltage level for at least a predetermined debounce time interval. The serial scan input logic 804 is coupled to programmable timers 806 which generate and indicate the end of the debounce delay interval initiated by the serial scan logic 804. Requiring the status to remain stable for the debounce time interval minimizes the inadvertent powering down of one of the hot-plug slots 36 due to a false value (i.e., a "glitch") indicated by one of the status signals STATUS[5:0]. When all of the status signals STATUS[5:0] remain at the same logical voltage level for at least the debounce time interval, the serial scan input logic 804 then proceeds to once again scan all thirty-two status signals STATUS[31:0] in the faster scan mode.

If the serial scan input logic 804 detects a change in one of the status signals STATUS[31:6], the serial scan input logic 804 instructs the timers 806 to measure another debounce delay interval, subsequently asserts the serial interrupt signal SI_INTR#, updates the interrupt register 800 with the signals STATUS[31:6] that have changed, and ignores further changes in the status signals STATUS[31:6] until the debounce time interval expires. After expiration of the debounce time interval, the serial scan input logic 804 proceeds to recognize changes in the thirty-two status signals STATUS[31:0].

When the serial interrupt signal SI_INTR# is asserted, the CPU 14 subsequently reads the interrupt register 800, determines which (may be more than one) status signals STATUS[31:0] caused the interrupt, and deasserts the serial interrupt signal SI_INTR# by writing a "1" to the bit or bits of the interrupt register 800 that have changed.

The CPU 14 may selectively mask interrupt requests caused by the status signals STATUS[31:0] by writing a "1" to a corresponding bit of a thirty-two bit interrupt mask register 810. The CPU 14 can also selectively read any byte of the status signals STATUS[47:0] by writing a byte number of the selected byte to a serial input byte register 812. The SIO circuit 50 then transfers the desired byte into a serial data register 815.

For example, to read the third byte (byte number two) of the status signals STATUS[23:16], the CPU 14 writes a "2" to the serial input byte register 812. The serial scan input logic 804 then serially shifts byte two of the status signals STATUS[23:16] into the serial data register 815. A busy status bit BS of the serial input byte register 812 is equal to "1" when the CPU 14 initially writes the desired byte number to the serial input byte register 812. The bit BS is cleared by the SIO circuit 50 after the requested byte has been shifted into the serial data register 815.

The CPU 14 can power up one of the slots 36 by writing a "1" to a corresponding bit of a slot enable register 817 and disable the slot 36 by writing a "0" to this bit. Furthermore, the CPU 14 can reset one of the slots 36 by writing a "1" to a corresponding bit of a slot reset register 819. The contents of the slot enable 817 and slot reset 819 registers are represented by signals SLOT_EN[5:0] and SLOT_RST_[5:0], respectively.

To initiate the request indicated by the slot enable 817 and reset 819 registers, the CPU 14 writes a "1" to an SO bit of control register 814. After the SO bit is 'asserted (which asserts, or drives high, a GO_UPDATE signal), the SIO circuit 50 initiates and controls the required power down and/or power up sequences.

NEW_CSID. One input of the AND gate 847 receives inverted bit enable signals BIT_ENABLE[5:0], and the other input of the AND gate 847 receives the signals SCAN_SW[5:0].

Only one of the bit enable signals BIT_ENABLE[5:0] is asserted at one time (when the scan state machine 840 is scanning), and the asserted bit indicates which one of the corresponding status signals STATUS[31:0] is represented by the signal NEW_CSID. Thus, when the scan state machine 840 is scanning, on every positive edge of the clock signal CLK, the signals SCAN_SW[5:0] are updated.

The bit enable signals BIT_ENABLE[31:0] are furnished by the output of a multi-bit multiplexer 832 that receives the bits BIT_ACTIVE[31:0] at its one input. The zero input of the multiplexer 832 receives a thirty-two bit signal indicative of logic zero. The select input of the multiplexer 832 receives the signal SHIFT_ENABLE.

For purposes of detecting a change in the status signals STATUS[5:0], a multi-bit, Exclusive Or (XOR) gate 848 furnishes switch change signals SW_CHG[5:0]. When one of the signals SW_CHG[5:0] is asserted, or high, the logical voltage of the corresponding status signal STATUS[5:0] changed during successive scans. One input of the XOR gate 848 is connected to the input of the flip-flop 836, and the other input of the XOR gate 848 receives the signals SCAN_SW [5:0].

As shown in Figure 31D, for purposes of indicating when the logical voltage level of a selected status signal STATUS [5:0] has remained at the logical voltage level for at least the duration of the debounce delay interval, the scan input logic 804 has six signals LSWITCH[5:0]. The non-inverting input of a D-type flip-flop 900 furnishes the signal LSWITCH [5] at its non-inverting output. The signal LSWITCH[5] is asserted, or driven high, to indicate the above-described condition and deasserted otherwise. The flip-flop 900 is clocked on the positive edge of the clock signal CLK, and the clear input of the flip-flop 900 receives the RST signal.

The input of the flip-flop 900 is connected to the output of a multiplexer 902 which furnishes a D_LSWITCH[5] signal. The select input of the multiplexer 902 is connected to the output of an AND gate 903 that receives a MAX5 signal and a SCAN_END signal. The SCAN_END signal, when asserted, indicates the scan state machine 840 has completed the current scan. Five signals (MAX5, MAX4, MAX3, MAX2, MAX1 AND MAX0) indicate whether the corresponding status signal STATUS[5], STATUS[4], STATUS[3], STATUS[2], STATUS[1], or STATUS[0], respectively, has remained at the same logical voltage level for a least the duration of the debounce time interval. The zero input of the multiplexer 902 receives the signal LSWITCH[5], and the one input of the multiplexer 902 receives the signal SCAN_SW[5]. The signal SCAN_END is furnished by the output of an AND gate 851 (Fig. 31B). The AND gate 851 receives a signal STOP_SCAN and a signal SCAN_DONE. The signal STOP_SCAN is asserted, or driven high, when conditions for ending the scanning by the scan state machine 840 are present, as further described below. The signal SCAN_END is a pulsed (for one cycle of the CLK signal) version of the STOP_SCAN signal. The signals LSWITCH [4]-LSWITCH[0] and D_LSWITCH[4]-D_LSWITCH[0] are generated in a similar fashion from the respective SCAN_SW [4]-SCAN_SW[0] signals and the respective signals MAX4-MAX0.

For purposes of updating the logical voltage level of the status signals STATUS[31:6] as these signals are scanned in, a multi-bit D-type flip-flop 905 (Fig. 31D) furnishes twenty-six signals SCAN_NSW[31:6]. One of the signals SCAN_NSW[31:6] is asserted, or driven high, to indicate this condition and deasserted otherwise. The flip-flop 905 is clocked on the positive edge of the clock signal CLK, and the clear input of the flip-flop 905 receives the RST signal.

The input of the flip-flop 905 is connected to the output of a multi-bit multiplexer 906. The select input of the multiplexer 906 receives an inverted CHECK_SWITCH_ONLY signal. The CHECK_SWITCH_ONLY signal is asserted, or driven high, when the scan state machine 840 is only scanning the status signals STATUS[5:0] or status signals STATUS[127:32] (i.e., ignoring changes in the signals STATUS[31:6]) and deasserted otherwise. The zero input of the multiplexer 906 receives the signals SCAN_NSW[31:6], and the one input of the multiplexer 906 is connected to the output of a multi-bit OR gate 907. One input of the OR gate 907 is connected to the output of a multi-bit AND gate 908, and the other 'input of the OR gate 907 is connected to the output of a multi-bit AND gate 872.

One input of the AND gate 908 receives the signals BIT_ENABLE[31:6]. The other input of the AND gate 908 is connected to the output of a multi-bit multiplexer 909. If the NEW_CSID signal is asserted, or high, the multiplexer 909 furnishes a twenty-six bit signal equal to "h3FFFFFF." Otherwise, the multiplexer furnishes a twenty-six bit signal equal to "0." One input of the AND gate 872 is connected to the inverted output of the AND gate 908, and the other input of the AND gate 872 receives the signals SCAN_NSW[31:6].

For purposes of storing the logical voltage level of the status signals STATUS[31:6] after every scan, a multi-bit, D-type flip-flop 871 furnishes twenty-six signals LNON_SW[31:6]. One of the signals LNON_SW[31:6] is asserted, or driven high, to indicate this condition and deasserted otherwise. The flip-flop 871 is clocked on the positive edge of the clock signal CLK, and the clear input of the flip-flop 871 receives the RST signal.

The input of the flip-flop 871 is connected to the output of a multi-bit multiplexer 870 which furnishes the signals D_LNON_SW[31:6]. The select input of the multiplexer 870 receives the signal SCAN_END. The zero input of the multiplexer 870 receives the signals LNON_SW[31:6], and the one input of the multiplexer 807 receives the signals SCAN_NSW[31:6].

As shown in Figure 31B, for purposes of generating the MAX0, MAX1, MAX2, MAX3, MAX4, and MAX5 signals,

815. The clear input of the flip-flop 912 receives the signal RST, and the flip-flop 912 is clocked on the positive edge of the CLK signal. The signal input of the flip-flop 912 is connected to the output of a multi-bit multiplexer 916. The select input of the multiplexer 916 is connected to the output of an AND gate 914, and the zero input of the multiplexer 916 receives the bits SI_DATA[7:0]. The AND gate 914 receives the signals GETTING_BYTE and SHIFT_ENABLE. Thus, when the serial scan logic 804 is not shifting in a requested byte of the status signals STATUS[47:0], the values of the bits SI_DATA[7:0] are preserved.

The one input of the multiplexer 916 is connected to the output of a multi-bit multiplexer 910. The one input of the multiplexer 910 is connected to the output of a multi-bit OR gate 911, and the zero input of the multiplexer is connected to the output of a multi-bit AND gate 915. The select input of the multiplexer 910 receives the signal NEW_CSID. One input of the AND gate 915 receives the bits SI_DATA[7:0], and an inverting input of the AND gate 915 is connected to the output of a 3X8 decoder 913. The decoder 913 receives the signal BIT[2:0]. One input of the OR gate 911 receives the bits SI_DATA[7:0], and the other input of the OR gate 911 receives the output of the decoder 913.

The serial input logic 804 furnishes five signals RST_SWITCH[5:0] (corresponding to the bit positions of the status signals STATUS[5:0]) to the ON/OFF control logic 820 which indicate, by their assertion, whether the corresponding slot 36a-f should be powered down. The ON/OFF control logic 820 indicates when the slot 36 (indicated by the RST_SWITCH[5:0] signals) has subsequently been powered down by the subsequent assertion of one of five signals CLR_SWITCH_[5:0] signals whose bit positions correspond to the signals RST_SWITCH[5:0]. After receiving the indication that the slot 36 has been powered down, the serial logic 804 then deasserts the corresponding RST_SWITCH [5:0] signal.

The signals RST_SWITCH[5:0] are furnished by the non-inverting output of a multi-bit, D-type flip-flop 891 (Fig. 31B). The clear input of the flip-flop 891 receives the reset signal RST, and the flip-flop 891 is clocked on the positive edge of the clock signal CLK. The input of the flip-flop 891 is connected to the output of a multi-bit OR gate 857 which has one input connected to the output of a multi-bit AND gate 859 and one input connected to the output of a multi-bit AND gate 855. One input of the AND gate 859 is connected to the output of a multiplexer 853, and the other input of the AND gate 859 receives latched slot enable signals LSLOT_EN[5:0] which indicate, by their assertion, whether the corresponding slot 36a-f is powered up. One input of the AND gate 855 receives the signals CLR_SWITCH_[5:0] signals. Another input of the AND gate 855 receives the signals RST_SWITCH[5:0]. Another input of the AND gate 855 is connected to the inverted output of the multiplexer 853.

The zero input of the multiplexer 853 receives a six bit signal indicative of zero. The one input of the multiplexer 853 is connected to the output of a multi-bit AND gate 849. One input of the AND gate 849 receives the signals D_LSWITCH[5:0], and the other input of the AND gate '849 receives the inverted signals L_SWITCH[5:0]. The select input of the multiplexer 853 receives the SCAN_END signal.

For purposes of generating the SI_INTR# signal, the serial scan logic 804 includes a D-type flip-flop 882 which furnishes the serial interrupt signal SI_INTR# at its inverting output. The flip-flop 882 is clocked on the positive edge of the CLK signal, and the clear input of the flip-flip 882 receives the RST signal. The input of the flip-flop 882 is connected to the output of an OR gate 881 which receives thirty two pending interrupt signals PENDING_IRQ[31:0], which indicate, by their assertion, or driving high, whether an interrupt is pending for the corresponding one of the status signals STATUS[31:0]. The signals PENDING_IRQ[31:0] are otherwise deasserted.

As shown in Figure 31E, a multi-bit, D-type flip-flop 979 furnishes the signals PENDING_IRQ[31:0] at its non-inverting output. The flip-flop 979 is clocked on the positive edge of the signal CLK signal and receives the signal RST at its clear input. The input of the flip-flop 979 is connected to the output of a multi-bit AND gate 981 which receives inverted interrupt mask signals INTR_MASK[31:0] at one input. The signals INTR_MASK[31:0] are indicative of corresponding bit of the interrupt mask register 810. The other input of the AND gate 981 is connected to the output of a multi-bit OR gate 835. One input of the OR gate 835 is connected to the output of a multi-bit AND gate 862, and the other input of the OR gate 835 is connected to the output of a multi-bit AND gate 834.

The AND gate 862 receives inverted PENDING_IRQ[31:0] signals and signals SET_PIRQ[31:0]. The signals SET_PIRQ[31:0] are asserted to indicate an interrupt request should be generated for the corresponding one of the status signals STATUS[31:0]. Therefore, the signals PENDING_IRQ[31:0] are updated with the signals SET_PIRQ[31:0] if not masked by the signals INTR_MASK[31:0].

The AND gate 834 receives the signals PENDING_IRQ[31:0], inverted signals SET_PIRQ[31:0] and inverted WR_INTR_REG[31:0] signals. The signals WR_INTR_REG[31:0] indicate the write data furnished by the CPU 14 to the interrupt register 800. The CPU clears an interrupt by writing a "1" to the corresponding bit of the interrupt register 800. Therefore, if this occurs, and no new interrupt requests are indicated for the corresponding one of the status signals STATUS[31:0], the corresponding one of the signals PENDING_IRQ[31:0] is cleared.

The signals SET_PIRQ[31:0] are furnished by the output of a multi-bit AND gate 839. One input of the AND gate 839 receives the signals UPDATE_IRQ[31:0]. The other input of the AND gate 839 is connected to the output of a multi-bit XOR gate 837. One input of the XOR gate 837 receives the signals D_INTR_REG[31:0], the other input of the XOR gate 837 receives the signals INTR_REG[31:0]. Therefore, when the bits of the interrupt register 800 transition

signal RST. If not idle, the ON/OFF state machine 998 controls one of three sequences: the power down sequence, the power on sequence, or the one pass sequence used to update the control signals POUT[39:0] as indicated by the slot enable 817 and LED control (not shown) registers. The ON/OFF state machine 998 asserts, or drives high, the load signal CSOLC_O for one cycle of the clock signal CLK of the register 80 until the ON/OFF state machine 998 determines the control signals POUT[39:0] are to be updated. When the control signals POUT[39:0] are updated, the ON/OFF state machine 998 negates the signal CSOLC_O which updates the control signals POUT[39:0].

The ON/OFF state machine 998 begins the power down sequence when either the software requests a power down of at least one of the slots 36, as indicated by the deassertion of the signals SLOT_EN[5:0]; or the serial scan input logic 804 determines at least one of the slots 36a-f should undergo the power down sequence, as indicated by the assertion of the signals RST_SWITCH[5:0]. To begin the power down sequence, the ON/OFF state machine 998 asserts the SO_UPDATE signal to begin a shifting phase and transitions from the IDLE state to a RSTON state.

During the RSTON state, the control logic 999 negates the reset signals RST#[5:0] for the slots 36 that are to be powered down, and the serial output logic 824 serially shifts the reset signals RST#[5:0] to the output register 80. The ON/OFF state machine 998 also negates the signal SO_UPDATE. Once all forty-control signals are shifted by the serial output logic 824 to the register 80, as indicated by the assertion of the signal SO_UPDATE_DONE, the ON/OFF state machine 998 transitions from the RSTON state to an OFF_ARB1 state.

In the OFF_ARB1 state, the ON/OFF state machine 998 requests control of the secondary PCI bus 32 by asserting the request signal CAYREQ#. The ON/OFF state machine 998 then transitions to an OFF_WGNT1 state where it waits for the grant of the secondary PCI bus 32. When the arbiter 124 grants control of the bus 32, as indicated by the assertion of the CAYGNT# signal, the ON/OFF state machine 998 negates the signal CSOLC_O for one cycle of the signal CLK to update the control signals POUT[39:0] and transitions to an OFF_LCLK1 state.

In the OFF_LCLK1 state, the ON/OFF state machine 998 asserts the signal SO_UPDATE to begin another shift phase. The ON/OFF state machine 998 transitions from the OFF_LCLK1 state to a bus off state BUSOFF. During the BUSOFF state, the control logic 999 deasserts, or drives high, the bus enable signals BUSEN#[5:0] for the slots 36 that are to be powered down, and the serial output logic 824 serially shifts the bus enable signals BUSEN#[5:0] to the output register 80. The ON/OFF state machine 998 also negates the signal SO_UPDATE. Once all forty-control signals are shifted by the serial output logic 824, as indicated by the assertion of the signal SO_UPDATE_DONE, the ON/OFF state machine 998 transitions from the BUSOFF state to an OFF_ARB2 state where the state machine 998 once again requests control of the secondary PCI bus 32. The state machine 998 then transitions to an OFF_WGNT2 state where it waits for the grant of the PCI bus 32. Once the grant is received, the state machine 998 transitions to an OFF_LCLK2 state where the control signals POUT[39:0] are updated by negating the signal CSOLC_O for one cycle of the signal CLK. The state machine 998 then transitions to a clock off state CLKOFF.

During the CLKOFF state, the control logic 999 deasserts, or drives high, the clock enable signals CLKEN#[5:0] for the slots 36 that are to be powered down. The bus enable signals BUSEN#[5:0] do not change, and the serial output logic 824 serially shifts the clock enable signals CLKEN#[5:0] to the output register 80. The ON/OFF state machine 998 also negates the signal SO_UPDATE. Once all forty control signals are shifted by the serial output logic 824, as indicated by the assertion of the signal SO_UPDATE_DONE, the ON/OFF state machine 998 transitions from the CLKOFF state to an OFF_ARB3 state, where the state machine 998 once again requests control of the PCI bus 32. The state machine 998 then transitions to an OFF_WGNT3 state where it waits for the grant of the PCI bus 32. Once the grant is received, the state machine 998 transitions to an OFF_LCLK3 state where the control signals POUT[39:0] are updated by negating the signal CSOLC_O for one cycle of the signal CLK. The state machine 998 then transitions to a power off state PWROFF.

During the PWROFF state, the control logic 999 deasserts, or dries low, the power enable signals PWREN[5:0] for the slots 36 that are to be powered down. The signals RST#[5:0], BUSEN#[5:0], and CLKEN#[5:0] do not change, and the serial output logic 824 serially shifts the power enable signals PWREN[5:0] to the output register 80. The ON/OFF state machine 998 also negates the signal SO_UPDATE. Once all forty control signals are shifted by the serial output logic 824, as indicated by the assertion of the signal SO_UPDATE_DONE, the ON/OFF state machine 998 transitions from the PWROFF state to an OFF_LCLK4 state where the signals POUT[39:0] are updated by negating the signal CSOLC_O for one cycle of the signal CLK. The state machine 998 then transitions to the IDLE state which completes the power down sequence.

If a power down sequence is not required, the ON/OFF state machine 998 then determines if the power up sequence is required. If either the software has requested at least one of the slots 36 to powered up or a power up of the expansion box 30 is pending, then the ON/OFF state machine 998 transitions from the IDLE state to a power on PWRON state to begin the power on sequence. To begin the power on sequence, the ON/OFF state machine 998 asserts the SO_UPDATE signal to begin a shift phase and transitions from the IDLE state to a power on state PWRON.

During the PWRON state, the control logic 999 asserts the power enable signals PWREN[5:0] for the slots 36 that are to be powered up, and the serial output logic 824 serially shifts the power enable signals PWREN[5:0] to the output register 80. The ON/OFF state machine 998 also negates the signal SO_UPDATE. Once all forty control signals are

The output shift state machine 920 furnishes an increment counter signal INC_CNTR to the bit counter 921. When the INC_CNTR signal is asserted, the bit counter 921 increments the value represented by the signal BIT_CNTR[5:0]. When a load counter signal LOAD_CNTR is asserted or when the RST signal is asserted, then the output of an OR gate 925, connected to a clear input of the bit counter 921, clears the signal BIT_CNTR[5:0].

The signal BIT_CNTR[5:0] is furnished to the select input of a multi-bit multiplexer 924 that furnishes the signal CSOD_O. The zero through eleven inputs of the multiplexer 924 receive the LED control signals LEDS[11:0]. The twelve through fifteen inputs of the multiplexer 924 receive general purpose output signals GPOA[3:0]. The sixteen through twenty-one inputs receive the reset signals RST#[5:0]. The twenty-two through twenty-seven inputs receive the clock enable signals CLKEN#[5:0]. The twenty-eight through thirty-three inputs receive the bus enable signals BUSEN#[5:0]. The thirty-four through thirty-nine inputs receive the power enable signals PWREN[5:0].

As shown in Figures 35A-B, the output shift state machine 920 enters an IDLE state when the signal RST is asserted. If the signal SO_UPDATE is asserted, then the output shift state machine 920 transitions from the IDLE state to a SHIFT1 state.

Because the output shift state machine 920 is clocked on the positive edge of the PCI clock signal CLK, the output shift state machine 920 transitions through a SHIFT1 state, a SHIFT2 state, a SHIFT3 state and a SHIFT4 state to generate the clock signal CSOSC_O that is one fourth of the frequency of the clock signal CLK. During the SHIFT1 and SHIFT2 states the clock signal CSOSC_O is negated, or low, and during the SHIFT3 and SHIFT4 states, the clock signal CSOSC_O is asserted, or high. When the current shift phase is completed, as indicated by the assertion of the signal MAXCNT, the shift state machine 920 returns to the IDLE state and the clock signal CSOSC_O is asserted until the beginning of the next shifting phase.

As shown in Figure 36, a HANG_PEND signal is received by the clear input of the register 80. The assertion, or driving high, or the HANG_PEND signal asynchronously clears the appropriate output control signals POUT[39:0] to power down all slots 36 when the PCI bus 32 is in a locked up condition, as further described below.

FAULT ISOLATION

The bus watcher 129 can detect for a hang condition on the secondary PCI bus 32. If a hang condition is detected, the bus watcher 129 sets a bus hang pending bit, which causes the SIO 50 to power down the slots on the secondary PCI bus 32 and a non-maskable interrupt (NMI) to be transmitted to the CPU 14. The CPU 14 responds to the NMI by invoking an NMI routine to isolate the slot(s) causing the hang condition. Once identified, the defective slot(s) are disabled or powered off.

For software diagnostic purposes, the bus watcher 129 in the downstream bridge chip 48 includes a bus history FIFO and a bus vector FIFO. When the secondary PCI bus 32 is functioning properly, the bus history information, which includes an address group (including the PCI address, PCI command signals, PCI master number, and the address parity bit) and a data group (including the PCI data, byte enable signals C/BE[3:0]_, parity error signal PERR_, the data parity bit, a burst cycle indication bit, and a data valid flag), are recorded by the bus watcher 129 in each transaction. When the PCI signal FRAME_ is asserted on the secondary PCI bus 32 to start a bus transaction, the address group and each subsequent data group are stored in the bus history FIFO. If the transaction is a burst transaction, then the burst cycle indication bit is set active on the second data phase. After the first data phase, the address field in the address group associated with the subsequent data groups in the burst transaction is incremented by 4 and the new address group and data group are stored in the next location of the bus history FIFO. If data is not transferred because of a retry condition or a disconnect-without-data condition, the valid data indication bit is set low.

Both the address group and the data group flow through a 2-stage pipeline to allow time for the data group to collect the data parity bit and data parity error bit, and stop the recording process when a data parity error occurs before the next address group is stored. If the bus hangs in the middle of a write data phase, the data is stored, and a bus-hang status bit is set in a bus-hang indication register 482 (Figure 42) accessible via configuration space. If the bus hangs in the middle of a read data phase, the data is marked as not valid, and the bus-hang bit is set.

Bus state vectors are assembled and stored in the bus vector FIFO, including the following PCI control signals: slot request signals REQ[7:0]_; slot grant signals GNT[7:0]_; the FRAME_ signal; the PCI device select signal DEVSEL_; the PCI initiator ready signal IRDY_; the PCI target ready signal TRDY_; the STOP signal; the PCI parity error signal PERR_; the PCI system error signal SERR_; and the LOCK_ signal. On each PCI clock in which the bus state vector changes, i.e., any one of the listed signals changes state, the new vector is stored into the bus vector FIFO.

The bus watcher 129 includes a watch-dog timer 454 (Figure 40) to determine whether the secondary bus 32 has locked up. If the watch-dog timer 454 expires, then the bus 32 has hung. The following are examples of bus-hang conditions that can be detected by the watch-dog timer 454: The FRAME_ signal is stuck high or low; the signal TRDY_ is not asserted in response to IRDY_; the PCI arbiter 124 does not grant the bus to any master; and a master requesting the bus 32 keeps getting retried.

When the watch-dog timer 454 expires, the bus hang pending bit is set active in the bus-hang indication register

is causing the failure.

If the variable I is determined 434 to be less than zero, then the handler checks 436 to determine if all populated slots have been enabled. If not, the variable N is incremented 437, the isolation-in-progress EV is updated 439, and the variable I is again set 441 equal to the value of N.

If the bus hang pending bit is set 438 active, then potentially two slots are disabled 440: slot N (which is the slot currently being enabled) and slot I (which is the slot currently being read from and written to). If the value of I and N are the same, then only slot N is disabled.

If the handler determines 436 that all populated slots have been enabled (and a failure could not be identified), then the handler logs 442 in the NVRAM its inability to isolate the failure. Next, the handler clears 428 the isolation-in-progress EV.

Referring to Figure 40, the watch-dog timer 454, provides output signals WD_TMR_OUT[17:0] (timer count value), HANG_PEND (bus hang condition present), EN_CAP (the software has enable capture of the bus and vector history information), TIME_OUT (the watch-dog timer 454 has timed out), a signal HANG_RCOVR_EN (set high by software to enable the hang recovery logic in the bus watcher 129 and in the SIO 50) and a signal CAP_ILLEG_PROT (to indicate an illegal cycle on the PCI bus 32).

The signal HANG_PEND is provided to the SIO 50 to shut down the secondary bus slots. The input signals to the watch-dog timer 454 include some of the PCI bus signals, a signal WRT_EN_CAP_1 (pulsed high by software to re-enable the capture of the bus history and bus vector information by the fault isolation block 129), and a power-good indicator signal SYNC_POWEROK (indicating that power in the computer system is stable).

A bus hang recovery state machine 456 receives the signals HANG_PEND, TIME_OUT, and HANG_RCOVR_EN from the watch-dog timer 454. The recovery state machine 456 also receives some of the PCI signals. The output signals from the bus hang recovery state machine 456 includes a device select signal DEVSEL_O for driving the PCI signal DEVSEL_, a signal STOP_O for driving the PCI signal STOP_, a signal SERR_EN which enables assertion of the system error signal SERR_, a signal BR_M_ABORT (indicating that the bus watcher 129 has recovered with a master abort), a signal BR_T_ABORT (indicating that the bus watcher 129 has recovered with a target abort), and a signal RCOVR_ACTIVE (for indicating when the bus hang recovery state machine 456 is active). The bus hang recovery state machine 456 ensures that the secondary PCI bus 32 is brought back to the idle state to allow the software to isolate the faulty slot. When the hang condition is detected, the SIO 50 powers down the secondary bus slots, which would automatically place the bus 32 into the idle state if one of the slot devices was the bus master when the hang condition occurred. However, if one of the slot devices was the target (and the bridge chip 48 was the master) when the bus hang occurred, then the bridge chip 48 would remain on the bus. To take the bridge chip off the bus, the recovery state machine 456 forces a retry cycle on the PCI bus 32 by asserting the signal STOP_.

A bus history capture block 458 monitors the PCI bus 32 for transactions, and presents the bus history information (including the address and data) on to output signals BUS_HIST_DATA3[31:0] (the bus history address), BUS_HIST_DATA2[31:0] (the bus history data), and BUS_HIST_DATA1[15:0] (parity error signal !PERR_, parity bit PAR, valid data bit VALID_DAT, address parity bit ADDRPAR, burst indicator BURST, master number MASTER[2:0], byte enable bits CBE[3:0]_, and command bits CMD[3:0]). The bus history capture block 458 asserts a signal HIST_RDY when data is available on the BUS_HIST_DATA signals, which is true at the end of each data phase in a normal transaction, or if the transaction is terminated with a master abort, a retry, while the assertion of the time out signal TIME_OUT.

A bus vector capture block 460 captures the states of certain PCI control signals when any of the those control signals changes state. The vector is captured and output as signals BUS_VECT_DATA[20:0], which contain the request signals !REQ[7:0]_, grant signals !GNT[7:0]_, time out signal TIME_OUT, lock signal LOCK_, system error signal SERR_, parity error signal PERR_, stop signal STOP_, target ready signal TRDY_, initiator ready signal IRDY_, device select signal DEVSEL_, and frame signal FRAME_. The bus vector capture block 460 asserts a signal VECT_RDY if any of the bus vector BUS_VECT_DATA[24:0] has changed or the watch-dog timer 454 has expired (TIME_OUT is high).

The bus history and bus vector signals are presented to the inputs of bus watcher FIFOs, which includes a 2-deep bus history FIFO and a 4-deep vector history FIFO. The outputs of the bus history FIFOs are presented as signals BUS_HIST_REG1[31:0], BUS_HIST_REG2[31:0], and BUS_HIST_REG3[31:0]. The outputs of the vector history FIFO are presented as signals BUS_VECT_REG[31:0]. The system software reads the outputs of the bus history FIFO by generating an I/O read cycle which causes a signal BUS_HIST_RD1 to be asserted, and reads the outputs of the vector FIFO by generating an I/O read cycle which causes a signal BUS_VECT_RD to be asserted.

Referring to Figure 41, the recovery state machine 456 begins in state IDLE when the signal SYNC_POWEROK is negated low, indicating that power is not yet stable. The state machine remains in state IDLE while the signal HANG_PEND is low. In state IDLE, signals BR_M_ABORT, BR_T_ABORT and RCOVR_ACTIVE are negated low. The signal RCOVR_ACTIVE is active high in the other states WAIT, ABORT, and PEND_OFF. If the signal SET_HANG_FEND is asserted high, the state machine transitions to state WAIT. In the transition, the signal

via the signal WRT_EN_CAP_1, it is maintained high. The signal CLR_EN_CAP is asserted to clear the signal EN_CAP (disable capture of information), which occurs when a time-out has occurred (TIME_OUT is high), a system error has occurred (SERR_ is low), a parity error has occurred (PERR_ is low), or an illegal bus protocol has been detected (CAP_ILLEG_PROT is high).

The signal CAP_ILLEG_PROT is generated by a D-type flip-flop 483, whose D input receives the output of an AND gate 485. One input of the AND gate receives the inverted state of the signal WRT_EN_CAP_1, and the other input receives the output of an OR gate 487. The OR gate 487 receives the signals CAP_ILLEG_PROT and SET_ILLEG_PROT. The signal SET_ILLEG_PROT is asserted when capture is enabled (EN_CAP is high), the state machine 456 is not active (RCOVR_ACTIVE is low), the bus is idle, and any of signals DEVSEL_, TRDY_, or IRDY_ is asserted low. This condition is an illegal condition, which triggers capture of the bus history and bus vector information.

Referring to Figure 43, the bus history ready signal HIST_RDY is generated by a D-type flip-flop 502, which is clocked by the signal PCLK and cleared by the signal RESET. The D input of the flip-flop 502 is connected to the output of an OR gate 504, whose inputs receive the signal TIME_OUT, a signal M_ABORT (master abort signal delayed by one PCLK), the output of an AND gate 506, and the output of an AND gate 508. The AND gate 506 asserts its output if a retry, disconnect C or target abort cycle is present on the secondary bus 32 (the signal FRAME_, the inverted state of the signal IRDY_, the inverted state of the signal STOP_, and the inverted state of the signal DSC_A_B are all true). The AND gate 508 asserts its output when a completed data transfer has occurred (the signals IRDY_ and TRDY_ are both low). Thus, the bus history information is loaded into the bus history FIFOs when the watch-dog timer 454 times out, a retry, disconnect C, or target abort condition is present, the master has aborted the cycle, or a cycle has successfully completed.

The valid data indication signal VALID_DAT is generated by a D-type flip-flop 510, which is clocked by the signal PCLK and cleared by the signal RESET. The D input of the flip-flop 510 is connected to the output of a NOR gate 512, which receives the signal TIME_OUT, master abort signal M_ABORT, and the output of the AND gate 506. Thus, data is valid unless a time out is detected, a master abort cycle is issued, or a retry, disconnect C, or target abort cycle is present.

The signal VECT_RDY is generated by a D type flip-flop 514, which is clocked by the signal PCLK and cleared by the signal RESET. The D input of the flip-flop 514 is connected to the output of an OR gate 516, which receives the time out signal TIME_OUT and a signal CHANGE_STATE indicating that one of the PCI control signals in the bus vector has changed state. Thus, the state vector information is loaded into the vector FIFOs whenever control signals on the PCI bus 32 change state or when a time-out occurred.

Referring to Figure 44, the bus history data {BUS_HIST_DATA3[31:0], BUS_HIST_DATA2[31:0], BUS_HIST_DATA1[15:0] is provided to the input of bus history register 540, which is the first stage of the bus history FIFO. The bus history 501 provides output signals BUS_HIST_FIFO1[79:0], to the register 542 (the second state of the pipeline), which provides output signals BUS_HIST_FIFO0[79:0]. Both bus history registers 540 and 542 are clocked by the signal PCLK and cleared when the power-good signal SYNC_POWEROK is low.

The bus history registers 540 and 542 are loaded when the output of an AND gate 518 is driven high. The AND gate 518 receives the enable capture bit EN_CAP and the OR of the bus history ready signal HIST_RDY and the CAP_ILLEG_PROT signal (OR gate 519). The output signals BUS_HIST_FIFO0[79:0] and BUS_HIST_FIFO1[79:0] are provided to the 0 and 1 inputs, respectively, of multiplexers 520, 522, and 524. Each of the multiplexers 520, 522, and 524 is selected by a read address signal HIST_FIFO_RD_ADDR (which starts out low to select the output of the bus register 502 and is toggled on each subsequent read). The multiplexers 520, 522, and 524 drive output signals BUS_HIST_REG3[31:0], BUS_HIST_REG2[31:0], and BUS_HIST_REG1[15:0], respectively.

The bus vector data signals BUS_VECT_DATA[24:0] are provided to the inputs of a bus vector register 544, whose output is routed to the input of a bus vector register 546. The output of the bus vector register 546 is routed to the input of a bus vector register 548, whose output is in turn routed to the input of a bus vector register 550. Each of the bus vector registers 0-3 are clocked by the signal PCLK and cleared when the signals SYNC_POWEROK is low. The bus vector registers are loaded when the output of the AND gate 521 is asserted high. The AND gate 521 receives the signal EN_CAP and the OR of signals VECT_RDY and CAP_ILLEG_PROT (OR gate 523). The bus vector registers 550, 548, 546 and 544 produce output signals BUS_VECT_FIFO0[24:0], BUS_VECT_FIFO1[24:0], BUS_VECT_FIFO2[24:0], and BUS_VECT_FIFO3[24:0], respectively, which are in turn provided to the 0, 1, 2, and 3 inputs of a multiplexer 526, respectively. The output of the multiplexer 526 provides signals BUS_VECT_REG[31:0], with the multiplexer 526 selecting one of its inputs based on the state of address signals VECT_FIFO_RD_ADDR[1:0] (which begin with a binary value 00 and is incremented on each successive read).

Thus, the bus history and bus state vector information is captured in response to assertion of signals HIST_RDY or VECT_RDY, respectively, or in response to assertion of the signal CAP_ILLEG_PROT if an illegal bus protocol condition is detected.

and if not, the CPU 14 increments 1010 the parameter CURRENT_PCI_BUS, as a slot 36 that is powered down or empty was found, and finds 1004 the next PCI-PCI bridge circuit or slot 36 that is powered down or empty. Thus, by incrementing 1010 the parameter CURRENT_PCI_BUS, the CPU 14 effectively reserves a bus number for the slot 36 that is powered down or empty. Alternatively, the CPU 14 may reserve more than one bus number for the slot 36 that is powered down or empty.

If the CPU 14 found a PCI-PCI bridge circuit, the CPU 14 then sets 1012 the primary bus number of the PCI-PCI bridge circuit equal to the parameter CURRENT_PCI_BUS. The CPU 14 then increments 1014 the parameter CURRENT_PCI_BUS and sets 1016 the secondary bus number of the PCI-PCI bridge equal to the new bus number indicated by the parameter CURRENT_PCI_BUS.

The CPU 14 then sets 1018 the subordinate bus number of the found PCI-PCI bridge circuit equal to the maximum possible number of PCI buses by writing to the subordinate bus number register 1218. This value for the subordinate bus number register 1218 is temporary and allows the CPU 14 to find and program additional downstream PCI-PCI bridge circuits or slots 36 that are powered down or empty.

The CPU 14 finds additional downstream PCI-PCI bridge circuits or slots 36 that are powered down or empty by preserving 1022 the parameters PCI_BUS, DEV and FCN and recursively calling 1022 the BUS_ASSIGN routine. The CPU 14 then restores 1024 the values for the parameters PCI_BUS, DEV and FCN, and returns the latest call of the BUS_ASSIGN routine to update the parameter CURRENT_PCI_BUS with the next PCI bus number to be assigned by the CPU 14.

The CPU 14 then updates 1026 the subordinate bus number of the found PCI-PCI bridge by setting 1026 the subordinate bus number equal to the parameter CURRENT_PCI_BUS. Thus, this completes the assignment of the PCI bus number to the found PCI-PCI bridge circuit and additional downstream PCI-PCI bridge circuits and slots 36 that are powered down or empty . The CPU 14 then finds 1004 the next PCI-PCI bridge circuit or slot 36 that is powered down or empty on the PCI bus indicated by the parameter PCI_BUS.

As shown in Figure 46, after the PCI bus numbers are assigned, the CPU 14 executes a memory space allocation routine called MEM_ALLOC to allocate memory space for PCI functions and slots 36 that are powered down or empty. The CPU 14 first initializes 1028 search parameters used in aiding the CPU 14 in finding the located PCI functions and slots 36 that are powered down or empty.

The CPU 14 then finds 1030 the next PCI function or slot 36 that is powered down or empty. If the CPU 14 determines 1032 that all PCI functions and all slots 36 that are powered down or empty have been allocated memory space, the CPU 14 returns from the routine MEM_ALLOC. Otherwise, the CPU 14 determines 1032 whether a PCI function was found.

If so, the CPU 14 allocates 1038 memory resources as specified by the PCI function. Otherwise, one of the slots 36 that is powered down or empty is found, and the CPU 14 allocates 1036 a default memory size and memory alignment for the slot 36. The default memory size can either be a predetermined size determined before power up of the computer system 10 or a size determined after a determination of the memory resources required by the computer system 10.

When allocating memory space, the CPU 14 programs memory base 1212 and memory limit 1214 registers of the PCI-PCI bridge circuits that are upstream of the found PCI function. The CPU 14 also programs base address registers of the corresponding PCI devices appropriately. The CPU 14 then finds 1030 the next PCI function or slot 36 that is powered down or empty.

As shown in Figure 47, after the PCI bus numbers are assigned, the CPU 14 executes an I/O space allocation routine called I/O_ALLOC to allocate I/O space for PCI functions and slots 36 that are empty. The CPU 14 first initializes 1040 search parameters used in aiding the CPU 14 in finding the located PCI functions and slots 36 that powered down or empty.

The CPU 14 finds 1042 the next PCI function or slot 36 that is powered down or empty. If the CPU 14 determines 1044 that all PCI functions and slots 36 that are powered down or empty have been allocated I/O space, the CPU 14 returns from the I/O_ALLOC routine. Otherwise, the CPU 14 determines 1044 whether a PCI function was found. If so, the CPU 14 allocates 1050 I/O resources as specified by the PCI function. Otherwise, a slot 36 that is powered down or empty was found, and the CPU 14 allocates 1048 a default I/O size and I/O alignment for the slot 36. The default I/O size can either be a predetermined size determined before power up of the computer system 10 or a size determined after a determination of the I/O resources required by the computer system 10.

When allocating I/O space, the CPU 14 programs the I/O base 1208 and limit 1210 registers of the PCI-PCI bridge circuits upstream of the PCI function or slot 36. The CPU 14 also programs base address registers of the corresponding PCI devices appropriately. The CPU 14 then finds 1042 the next PCI function or slot 36 that is powered down or empty.

As shown in Figure 48, after initial configuration, when an interrupt is generated that indicates one of the levers 802 has opened or closed, the CPU 14 executes an interrupt service routine called CARD_INT. The CPU 14 reads 1052 the contents of the interrupt register 800 to determine 1053 whether the lever 802 has been opened or closed. If the CPU 14 determines 1053 that the lever 802 causing the interrupt was opened, the CPU 14 returns from the routine CARD_INT.

CFGCMD, ADDR01, and the inverted state of the signal UPSTREAM_CHIP.

The bridge chip receiving the type 0 transaction uses the register number field 250 in the configuration transaction address to access the appropriate configuration register. The function number field 252 specifies one of eight functions to be performed in a multi-functional device during the configuration transaction. A PCI device can be multi-functional and have such functions as a hard disk drive controller, a memory controller, a bridge, and so forth.

When the bridge chip 26 sees a type 1 configuration transaction on its upstream bus 26, it can forward the transaction either downstream, translate the transaction to a type 0 transaction, convert the transaction to a special cycle, or ignore the transaction (based on the bus number parameters stored in the configuration registers 105 or 125). If a transaction is forwarded, it is up to the PCI master of the destination bridge chip to convert the type 1 transaction to the corresponding appropriate transaction. If a bridge chip handles the transaction itself, then it responds by asserting the signal DEVSEL_ on the PCI bus and handles the transaction as a normal delayed transaction.

In a type 1 configuration transaction, the bus number field 260 selects a unique PCI bus in the PCI hierarchy. PCI target block 103 passes a type 1 configuration cycle from the upstream chip 26 down to the downstream bridge chip 48 if a signal PASS_TYP1_DS is asserted by an AND gate 284. The AND gate 284 receives the signal TYP1_CFG_CYC_US and a signal IN_RANGE (the bus number field 260 is greater than or equal to the stored secondary bus number and less than or equal to the stored subordinate bus number). The other input of the AND gate 284 is connected to the output of an OR gate 286, which has one input connected to the output of an AND gate 288 and the other input receiving the inverted state of a signal SEC_BUS_MATCH. Thus, if a type 1 cycle is detected, the signal IN_RANGE is asserted and the bus number field 260 does not match the stored secondary bus number, the signal PASS_TYP1_DS is asserted. If the bus field 260 does not match the stored secondary bus number, then bus devices on or below the downstream bus 32 are addressed. The AND gate 288 asserts its output high if the signal SEC_BUS_MATCH is asserted high and the device number field 258 indicates that the target of the type 1 configuration cycle is the configuration space of the downstream bridge chip 48. If this is true, the type 1 configuration transaction is forwarded down the cable 28 to the downstream bridge chip 48 for translation to a type 0 configuration transaction. The PCI target 121 in the downstream bridge chip 48 responds to the transaction and reads and writes the downstream bridge configuration registers 125 according to the type 0 transaction. The control pins of the downstream chip are driven and read and write data appear on the downstream PCI bus 32 as if a type 0 transaction is running on the downstream bus (for debug purposes), although each IDSEL on the downstream bus 32 is blocked so that no device actually responds to the type 0 transaction.

If the PCI target block 103 in the upstream bridge chip 26 detects a type 1 configuration transaction on its upstream bus 24, having a bus number field equal to the stored secondary bus number (the cable bus 28) but not addressing device 0 (searching for other devices on the cable bus 28), then the target block 103 ignores the transaction on the primary bus 26.

If the PCI target 121 detects a type 1 configuration write transaction (WR_ high) on the secondary PCI bus 32, which has a bus number field 260 outside the range of the secondary bus number and subordinate bus number (IN_RANGE low), and if the device number 258, the function number 256, and the register number 254 indicate a special cycle (SP_MATCH high), then a signal PASS_TYP1_US is asserted by an AND gate 290. The AND gate 290 receives the signal TYP1_CFG_CYC_DS, the signal SP_MATCH, the write/read strobe WR_, and the inverted state of the signal IN_RANGE. When the PCI master 101 in the upstream bridge chip 26 receives such a cycle, it runs a special cycle on the primary PCI bus 24.

Configuration transactions are ignored by a bridge chip under certain conditions. If the target block 103 in the upstream bridge chip 26 detects a type 1 configuration transaction on the PCI bus 24 (its upstream bus), and the bus number field 260 is less than the secondary bus number or greater than the subordinate bus number stored in the bridge chip's configuration space, then the target block 103 ignores the transaction.

If the target block 121 in the downstream bridge chip 48 detects a type 1 configuration transaction on the secondary PCI bus 32 (its downstream bus), and the bus number field 260 is greater than or equal to the secondary bus number and less than or equal to the subordinate bus number stored in the bridge chip's configuration space, then the target block 121 ignores the transaction. In addition, type 1 configuration commands going upstream are ignored if the type 1 command does not specify a conversion to a special cycle transaction regardless of the bus number specified in the type 1 command.

Referring to Figure 53B, the PCI master 101 or 123 watches for a configuration cycle transferred over the cable 28. If the PCI master 123 in the downstream bridge chip 48 detects a type 1 configuration transaction from the upstream bridge chip 26, it compares the bus number field 260 with the primary bus number and secondary bus number stored in the configuration space of the bridge chip 48. If the bus number field 260 matches either the stored primary bus number (i.e., cable 28) or the stored secondary bus number (addressing a device directly connected to the downstream bus 32), the downstream bridge chip 48 translates the transaction to a type 0 transaction (by setting AD[1:0] = 00) as it passes the configuration transaction onto the bus. The type 0 transaction is performed on the PCI bus 32 by the PCI master block 123.

which the various parameters are to be calculated. The timer 1300 is programmed to the hexadecimal value FFFFFFFF. If the PCI clock PCICLK2 is running at 33 MHz, then the timer period is approximately 2 minutes. When the timer 1300 decrements to 0, it asserts a signal GL_TIME_EXPIRE.

The bus monitor 127 includes 7 slot-specific bus-busy counters 1302A-G, six of the counters corresponding, respectively, to the 6 slots on the secondary PCI bus 32 and one to the SIO 50. The bus-busy counters 1302A-G are cleared when the signal GL_TIME_EXPIRE is asserted. Depending on which bus device has control of the secondary bus 32, the bus-busy counter 1302 increments on every PCI clock in which the secondary PCI bus FRAME_ or IRDY_ signal is asserted. The appropriate one of the seven counters is selected by one of the grant signals GNT[7:1]_. Thus, for example, the bus-busy counter 1302A is selected when the signal GNT[1]_ is asserted low, indicating that the SIO is the current master on the secondary PCI bus 32.

Seven data-cycle counters 1306A-G, corresponding, respectively, to the 6 slots on the secondary PCI bus 32 and the SIO 50, keep track of the time during which a data transfer is actually occurring between a master and a target during a transaction on the PCI bus 32. The selected data-cycle counter 1306 is incremented on every PCI clock in which the secondary bus IRDY_ and TRDY_ signals are both asserted low. The data-cycle counters 1306A-G are cleared when the signal GL_TIME_EXPIRE is asserted.

Six DCQ data counters 1310A-F are included in the bus monitor 127 for keeping track of the amount of data loaded into the DCQ buffers. The six DCQ data counters 1310A-F correspond to the 6 slots on the secondary PCI bus 32. The selected DCQ data counter 1310 increments on every PCI clock in which delayed read completion (DRC) data is received from the cable 28 and loaded into the prefetch buffers.

Another set of counters, DCQ-data-used counters 1314A-F, are used to keep track of the amount of data loaded into the DCQ 144 actually used by the 6 slots on the secondary PCI bus 32. The selected DCQ-data-used counter 1314 increments on every PCI clock in which the secondary bus master reads data from the corresponding DCQ buffer. Both the DCQ-data counters 1310A-F and DCQ-data-used counters 1314A-F increment on each data cycle regardless of the number of bytes actually transferred. In most cases, the number of bytes transferred in each data cycle is 4.

When the global period timer 1300 times out and asserts the signal GL_TIME_EXPIRE, several events occur. First, the global period timer 1300 reloads its original count value, which is the hexadecimal value FFFFFFFF. The contents of all the other counters, including the bus-busy counters 1302A-G, data-cycle counters 1306A-G, DCQ data counters 1310A-F, and DCQ-data-used counters 1314A-F, are loaded into registers 1304, 1308, 1312, and 1316, respectively. The counters 1302, 1306, 1310, and 1314 are then cleared to 0. The global period timer 1300 then begins to count again after it is reloaded with its original value.

The signal GL_TIME_EXPIRE is provided to the interrupt receiving block 132, which forwards the interrupt over the cable 28 to the interrupt output block 114, which in turn generates an interrupt to the CPU 14. The CPU 14 responds to the interrupt by invoking an interrupt handler to perform the bus performance analysis. The interrupt handler accesses the contents of the registers 1304, 1308, 1312, and 1316, and calculates the several parameters, including the bus utilization, bus efficiency, and prefetch efficiency parameters associated with the 6 secondary bus slots and the SIO 50.

The bus utilization parameter is the value of the bus-busy counter 1302 divided by the initial value of the global period timer 1300, which is the hexadecimal value FFFFFFFF. Thus, bus utilization is the percentage of the total global time during which a bus master is performing a bus transaction.

A PCI transaction includes an address phase and at least one data transfer phase. A bus master asserts the signal FRAME_ to indicate the beginning and duration of an active bus transaction. When the signal FRAME_ is deasserted, that indicates the transaction is in the final data phase or the transaction has been completed. The signal IRDY_ indicates that the bus master is able to complete the current data phase of the bus transaction. During a write, the signal IRDY_ indicates that valid data is present on the bus. During a read, the signal IRDY_ indicates the master is prepared to accept read data. The addressed PCI target responds to the bus transaction by asserting the signal TRDY_ to indicate that the target is able to complete the current data phase of the transaction. During a read, the signal TRDY_ indicates that valid data is present on the bus; during a write, the signal TRDY_ indicates the target is prepared to accept data. Wait states can be inserted between the address and data phases and between consecutive data phases of the bus transaction. During the address phase or the wait states, no data transfer is actually occurring.

Actual data transfer is occurring only when both signals IRDY_ and TRDY_ are asserted low. To determine the data transfer bus efficiency, the interrupt handler divides the value of the data-cycle counter 1306 by the value of the bus-busy counter 1302. The bus efficiency represents the amount of time during which a data transfer actually occurs during a bus transaction. By calculating this value, the computer system can be made aware of target devices which require many wait states and therefore are inefficient.

The bridge chip 48 can fetch data from the primary PCI bus 26 and store the data in the DCQ 144. The DCQ 144 has eight buffers, each being assignable to a secondary bus master. For example, a memory read multiple transaction generated by a secondary bus master targeted at the primary bus will cause bridge 26, 48 to fetch 8 cache lines from the memory 20 and load into the DCQ 144. A memory read line transaction will cause the PCI-PCI bridge 26, 48 to fetch one line of data from the memory 20. In addition, as described in conjunctin with Figures 75 and 79, the PCI-PCI

1729 and transfers the read contents to a six bit I_2O subordinate register 1728 (Fig. 91) of the bridge chip 48. The register 1728 indicates the subordinate status (I_2O processor 1700 subordinate or CPU 14 subordinate) of the bus devices in the same manner as the register 1729. Before the CPU 14 writes to the register 1728, the register 1728 contains all ones (value at power up) which allows the CPU 14 to scan the bus 32 for the I_2O processor 1700. The interrupt receiving block 132 uses the register 1728 to identify which interrupt requests received by the block 132 should be routed to the CPU 14 and which interrupt requests received by the block 132 should be routed to the I_2O processor 1700 for processing. Furthermore, the logic 1710 uses the contents of the register 1728 to block recognition by the CPU 14 of the I_2O subordinate devices 1701-1702 from the CPU 14.

For purposes of indicating to the interrupt receiving block 132, which bus device, if any, is an I_2O processor, the CPU 14 sets one bit of an I_2O slot register 1730 (Fig. 92) whose bits 0-5 correspond to the slots 36a-f, respectively. For this register 1730, located inside the bridge chip 48, a value of "0" for a bit indicates the associated slot 36 does not have an I_2O processor and a value of "1" for the bit indicates the associated slot 36 has an I_2O processor.

As shown in Figure 90, the logic 1710 includes a multi-bit AND gate 1711 which furnishes signals AD_IDSEL[5:0] to address/data lines of the bus 32 to select devices on the bus 32 during configuration cycles. The AND gate 1711 receives a six bit signal ENABLE[5:0] having bits indicative of and corresponding to bits of the I_2O subordinate register 1728. The AND gate 1711 also receives the typical identification select signals SLOT_IDSEL[5:0] furnished by the bridge chip 48 for selecting devices on the bus 32 during configuration cycles. Therefore, the signals ENABLE[5:0] are used to selectively mask the signals SLOT_IDSEL[5:0] from the PCI bus 32-when configuration cycles are run by the CPU 14.

For purposes of controlling the destination of interrupt requests from the slots 36a-d, the four standard PCI interrupt request signals (INTA#, INTB#, INTC# and INTD#) provided by each slot 36 are furnished to multiplexing circuitry 1712 (Fig. 88). The multiplexing circuitry 1712 serializes the PCI interrupt request signals received from the slots 36 and furnishes the signals to the interrupt receiving block 132 via four time multiplexed serial interrupt request signals: INTSDA#, INTDSB#, INTSDC#, and INTSDD#.

As shown in Figure 89, the interrupt receiving block 132 furnishes interrupt request signals for the CPU 14 to the interrupt output block 114 via a time multiplexed serial interrupt request signal INTSDCABLE#. The interrupt receiving block 132 furnishes interrupt request signals for the I_2O processor 1700 via a time multiplexed serial interrupt request signal INTSDIIO# furnished via a PCI INTC# line 1709 of the bus 32 to the I_2O processor 1700

The interrupt output block 114 furnishes the interrupt requests destined for the CPU 14 to one or more of the standard PCI interrupt request lines (INTA#, INTB#, INTC#, and INTD#) of the PCI bus 24. An interrupt controller 1900, external to the bridge chip 26, receives the interrupt requests from the PCI interrupt request lines of the PCI bus 24. The interrupt controller 1900 prioritizes the interrupt requests (which may include interrupt requests from other devices on the PCI bus 24) and furnishes them to the CPU 14. The interrupt output block 114 may either asynchronously (when in an asynchronous mode) furnish the interrupt request signals to the interrupt request lines of the PCI bus 24 or serially (when in a serial mode) furnish the interrupt request signals to the INTA# line of the PCI bus 24, as further described below.

As shown in Figure 95, all of the time multiplexed serial data signals represent their data via an interrupt cycle 1850 which comprises eight successive time slices (T0-T7). The duration of each time slice is one cycle of the PCI clock signal CLK. Each time slice represents a "snapshot" of the status of one or more interrupt request signals. As shown in Figure 99, the signal INTSDA# represents the sampled INTA# interrupt request signals from the slots 36a-f. The signal INTSDB# represents the sampled INTB# interrupt request signals from the slots 36a-f. The signal INTSDC# represents the sampled INTC# interrupt request signals from the slots 36a-f. The signal INTSDD# represents the sampled INTD# interrupt request signals from the slots 36a-f.

For purposes of combining the interrupt signals INTSDA#-D# into the signal INTSDIIO#, the interrupt receiving block 132 logically ANDs the signals INTSDA#-D# together while simultaneously masking interrupt request signals destined for the CPU 14. Similarly, for purposes of combining the interrupt signals INTSDA#-D# into the signal INTSDCABLE#, the interrupt receiving block 132 logically ANDs the signals INTSDA#-D# together while simultaneously masking interrupt request signals destined for the CPU 14.

For the purpose of instructing the interrupt output block 114 when another interrupt cycle 1850 is beginning, the interrupt receiving block 132 furnishes a synchronization signal INTSYNCCABLE# to the interrupt output block 114. The falling, or negative, edge of the signal INTSYNCCABLE# indicates that time slice TO of the interrupt cycle 1850 transmitted via the signal INTSDCABLE# is beginning on the next positive edge of the CLK signal. A signal INTSYNCIIO# is used in an analogous fashion to indicate an upcoming time slice T0 of the interrupt cycle 1850 transmitted via the signal INTSDIIO#. The signal INTSYNCIIO# is furnished by the interrupt receiving block 132 to the I_2O processor 1700 via a PCI INTD# line 1713 of the bus 32. For the purpose of instructing the multiplexing circuitry 1712 when to transmit another interrupt cycle 1850 via the interrupt signals INTSDA#-D#, the interrupt receiving block 132 furnishes a synchronization signal INTSYNC# to the multiplexing circuitry 1712. The falling, or negative, edge of the signal INTSYNC# indicates the muliplexing circuitry 1712 should being transmitting time slice T0 of the signals INTSDA#-D#

signal. The AND gate 1782 receives an inverted CAY_INT signal and the G_CNTRL signal.

The outputs of the AND gates 1772-1782 are connected as inputs to an OR gate 1784 which has its output connected to the signal input of a D-type flip-flop 1786. The flip-flop 1786 is clocked on the positive edge of the CLK signal, and the set input of the flip-flop 1786 receives the RST signal. The inverting output of the flip-flop 1786 furnishes the INTSDCABLE# signal.

Four AND gates 1790-1796 are used to combine the INTSDA#-D# signals and mask selected interrupt request signals from the I_2O processor 1700. The AND gate 1790 receives an inverted INTSDA# signal and an inverted MASKA signal. Another input of the AND gate 1790 is connected to the output of a NOR gate 1802 which masks the INTSDA# signal during the time slices T0 and T7 because no card interrupt requests are include in these time slices. The NOR gate 1802 receives the bits G_CNTRL[0] and G_CNTRL[7]. The AND gate 1792 receives an inverted INTSDB# signal and an inverted MASKB signal. Another input of the AND gate 1792 is connected to the output of a NOR gate 1804 which masks the INTSDB# signal during the time slices T1 and T4 because no card interrupt requests are include in these time slices. The NOR gate 1802 receives the bits G_CNTRL[1] and G_CNTRL[4].

The AND gate 1794 receives an inverted INTSDC# signal and an inverted MASKC signal. Another input of the AND gate 1794 is connected to the output of a NOR gate 1806 which masks the INTSDC# signal during the time slices T2 and T5 because no card interrupt requests are include in these time slices. The NOR gate 1806 receives the bits G_CNTRL[2] and G_CNTRL[5]. The AND gate 1796 receives an inverted INTSDD# signal and an inverted MASKD signal. Another input of the AND gate 1796 is connected to the output of a NOR gate 1808 which masks the INTSDD# signal during the time slices T3 and T6 because no card interrupt requests are include in these time slices. The NOR gate 1808 receives the bits G_CNTRL[3] and G_CNTRL[6].

The outputs of the AND gates 1790-1796 are connected as inputs to an OR gate 1798 which has its output connected to the signal input of a D-type flip-flop 1800. The flip-flop 1800 is clocked on the positive edge of the CLK signal, and the set input of the flip-flop 1800 receives the RST signal. The inverting output of the flip-flop 1800 furnishes the INTSDIIO# signal.

As shown in Figure 98, the interrupt output block 114 includes a three bit counter 1820 of common design with the counter 1745. The counter 1820 is clocked on the positive edge of the signal CLK, furnishes an output signal G_CNTR2 [2:0], and begins counting from zero to seven after being reset by the INTSYNC# signal.

For purposes of furnishing the INTSYNCCPU# signal, the interrupt output block 114 includes a D-type flip-flop 1822 that is clocked on the positive edge of the CLK signal. The set input of the flip-flop 1822 receives the RST signal, and the signal input of the flip-flop 1822 receives the INTSYNCCABLE# signal. The non-inverting output of the flip-flop 1822 furnishes the INTSYNCCPU# signal.

For purposes of furnishing the INTSDCPU# signal, the interrupt output block 114 includes a D-type flip-flop 1824 that is clocked on the positive edge of the CLK signal. The set input of the flip-flop 1824 receives the RST signal, and the signal input of the flip-flop 1824 receives the INTSDCABLE# signal. The non-inverting output of the flip-flop 1824 furnishes the INTSDCPU# signal.

The interrupt requests received by the interrupt receiving block 114 are furnished to the interrupt controller 1900 either asynchronously or serially. In the asynchronous mode, the interrupt requests are mapped to the four PCI interrupt lines (commonly referred to as a "barber poling") on the PCI bus 24 as shown in Figure 100.

For purposes of holding the interrupt information provided by the INTSDCABLE# signal, the interrupt output block 114 includes an eight bit register 1826. All signal inputs receive the INTSDCABLE# signal. The load enable inputs of bits 0-7 receive the bits G_CNTR[0]-G_CNTR[7], respectively. Therefore, for example, during time slice T4, bit 3 is loaded with the value represented by the INTSDCABLE# signal. Bits 0 (represented by a INT_A1 signal) and 4 (represented by a INT_A2 signal) are mapped into a CPUINTA# signal. Bits 1 (represented by a INT_B1 signal) and 5 (represented by a INT_B2 signal) are mapped into a CPUINTB# signal. Bits 2 (represented by a INT_C1 signal) and 6 (represented by a INT_C2 signal) are mapped into a CPUINTC# signal. Bits 3 (represented by a INT_D1 signal) and 7 (represented by a INT_D2 signal) are mapped into a CPUINTD# signal.

Four OR gates 1828-1834 furnish the signals CPUINTA#, 'CPUINTB#, CPUINTC#, and CPUINTD#, which are provided to the PCI interrupt lines INTA#, INTB#, INTC# and INTD#, respectively, of the PCI bus 24. The OR gate 1828 has one input connected to the output of an AND gate 1836. The AND gate receives an inverted CM signal. The signal CM is furnished by a bit of a configuration register of the bridge chip 26 and is asserted, or driven high, to indicate the asynchronous mode and deasserted, or driven low, to indicate the synchronous mode. The AND gate 1836 also receives the signal INT_A1, the signal INT_A2, and a signal ECC_ERR_UP (used to indicate an error in cable transmissions).

The OR gate 1828 has an input connected to the output of an AND gate 1838. The AND gate 1838 receives the CM signal and the INTSDCPU# signal. Another input of the AND gate 1838 is connected to the output of an OR gate 1848. The OR gate 1848 receives the ECC_ERR_UP signal and the bit G_CNTR2[0].

The OR gate 1830 has one input connected to the output of an AND gate 1840 and one input connected to the output of an AND gate 1842. The AND gate 1840 receives an inverted CM signal, the signal INT_B1, and the signal

a clamp configured to selectively prevent removal of the circuit card from the connector when the clamp is engaged; and
circuitry connected to monitor the engagement status of the clamp, and to regulate delivery of power to the connector based on the engagement state of the clamp.

12. The system of claim 11, further comprising:

a communication link, and
wherein the circuitry is further connected to regulate coupling of a communication link to the connector based on the engagement state of the clamp.

13. The system of claim 11 or claim 12, wherein the circuitry includes:

a switch actuated by the clamp furnishing an indication of the position of the switch; and
a circuit connected to update the engagement status when the indication indicates the same position for a predetermined duration.

14. A computer system comprising:

a central processing unit; and
a system according to any of claims 11 to 13.

15. The computer system of claim 14, wherein the circuit is configured to provide an interrupt request to the central processing unit to indicate when the engagement status changes.

16. The computer system of claim 14 or claim 15, wherein the circuit includes a buffer, accessible by the central processing unit, connected to store the indication of the engagement status.

17. The computer system of any of claims 14 to 16, wherein the circuit monitors a power status signal of the connector and provides an indication of the power status signal to the central processing unit.

18. The computer system of any of claims 14 to 17, further comprising:
a circuit, responsive to the circuit connected to monitor the engagement status of the clamp, connected to furnish power to the connector when the clamp is engaged and to remove power from the connector when the clamp is not engaged.

19. The computer system of claim 18, wherein the central processing unit selectively enables the connector to receive power, the computer system further comprising:

a buffer connected to indicate if the central processing unit has enabled the connector to receive power, and
wherein the circuit connected to furnish power only provides power if the central processing unit has enabled the connector to receive power.

20. The computer system of claim 18 or claim 19, further comprising:
a bus.

21. The computer system of claim 20, further comprising:
a circuit, responsive to the circuit for monitoring, for coupling the bus to the connector when the clamp is engaged and electrically isolating the bus from the connector when the clamp is not engaged.

22. The computer system of claim 21, wherein the central processing unit selectively enables the bus to be coupled to the connector, the computer system further comprising:

a buffer connected to indicate if the central processing unit has enabled the bus to be connected to the connector, and
wherein the circuit connected to furnish power only couples the bus to the connector if the central processing unit has enabled the connector to be coupled to the bus.

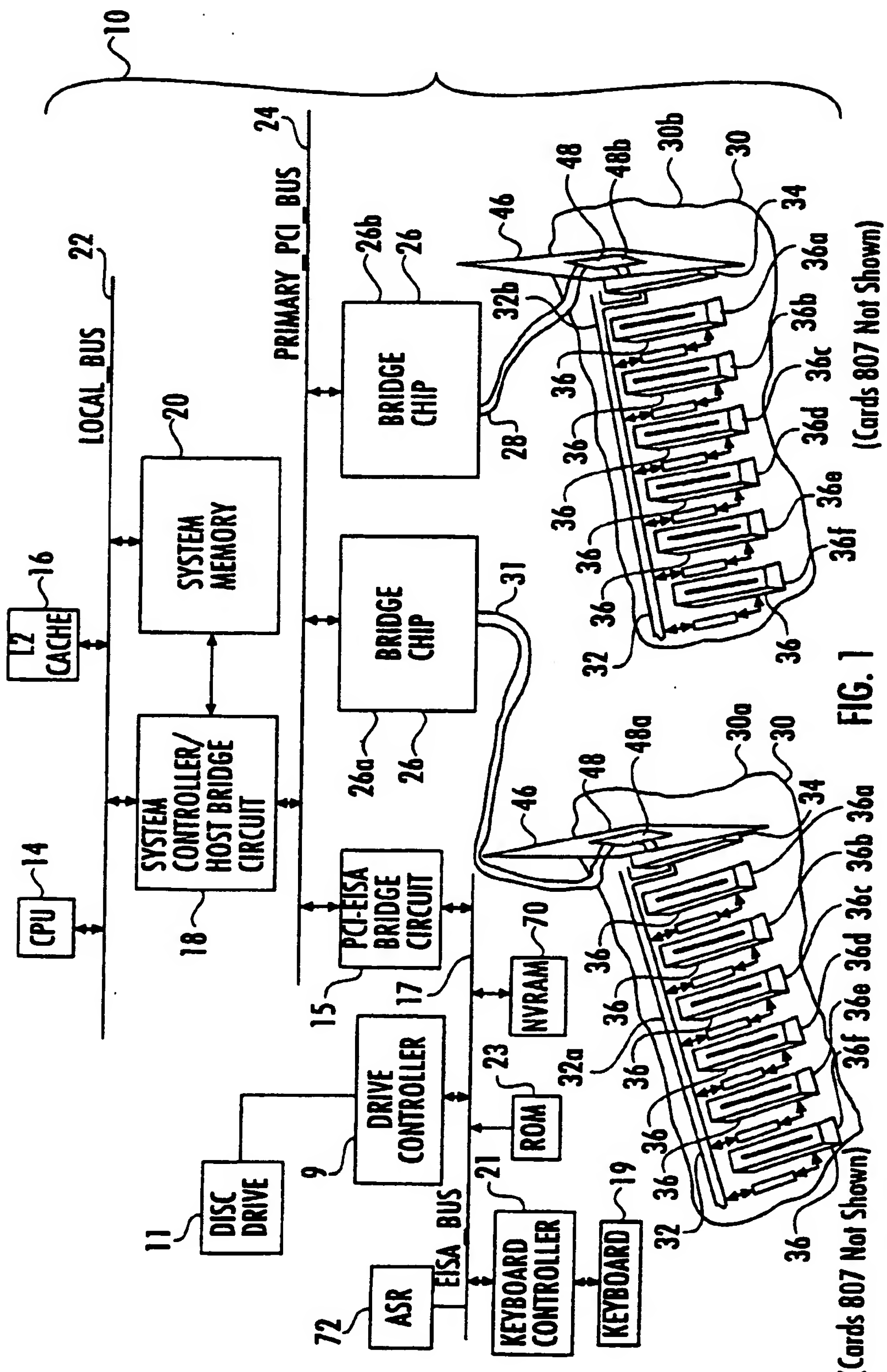

CPU
L2 CACHE
LOCAL_BUS
SYSTEM CONTROLLER/ HOST BRIDGE CIRCUIT
SYSTEM MEMORY
PRIMARY_PCI_BUS
BRIDGE CHIP
BRIDGE CHIP
PCI-EISA BRIDGE CIRCUIT
NVRAM
ROM
DRIVE CONTROLLER
DISC DRIVE
EISA_BUS
KEYBOARD CONTROLLER
KEYBOARD
ASR
(Cards 807 Not Shown)
(Cards 807 Not Shown)

FIG. 1

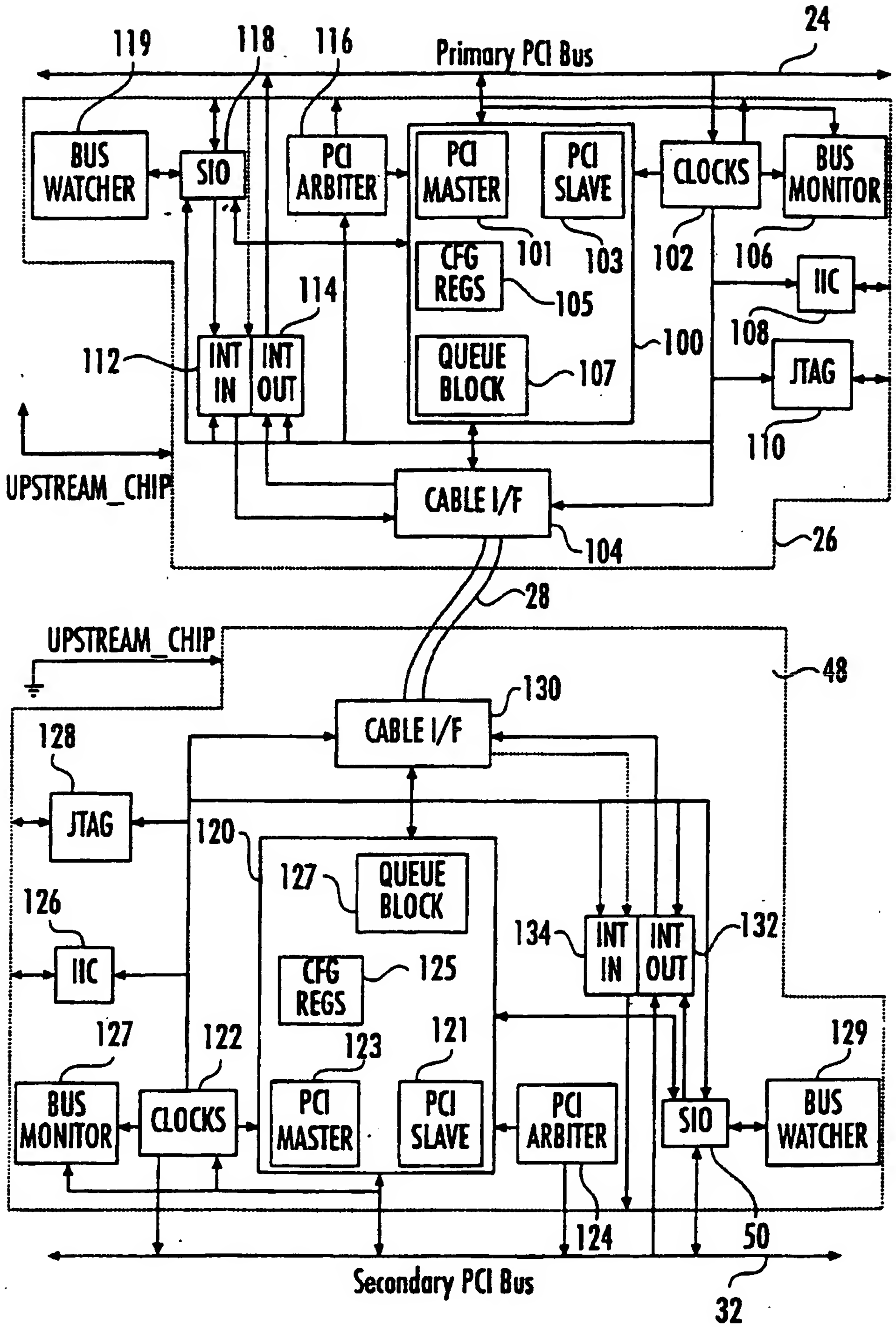

119
118
116
Primary PCI Bus
24
BUS WATCHER
SIO
PCI ARBITER
PCI MASTER
PCI SLAVE
CLOCKS
BUS MONITOR
CFG REGS
101
103
102
106
IIC
114
105
108
112
INT IN
INT OUT
QUEUE BLOCK
107
100
JTAG
110
UPSTREAM_CHIP
CABLE I/F
104
26
28
UPSTREAM_CHIP
48
130
CABLE I/F
128
JTAG
120
127
QUEUE BLOCK
126
134
INT IN
INT OUT
132
IIC
CFG REGS
125
127
122
123
121
129
BUS MONITOR
CLOCKS
PCI MASTER
PCI SLAVE
PCI ARBITER
SIO
BUS WATCHER
124
50
Secondary PCI Bus
32

FIG. 3

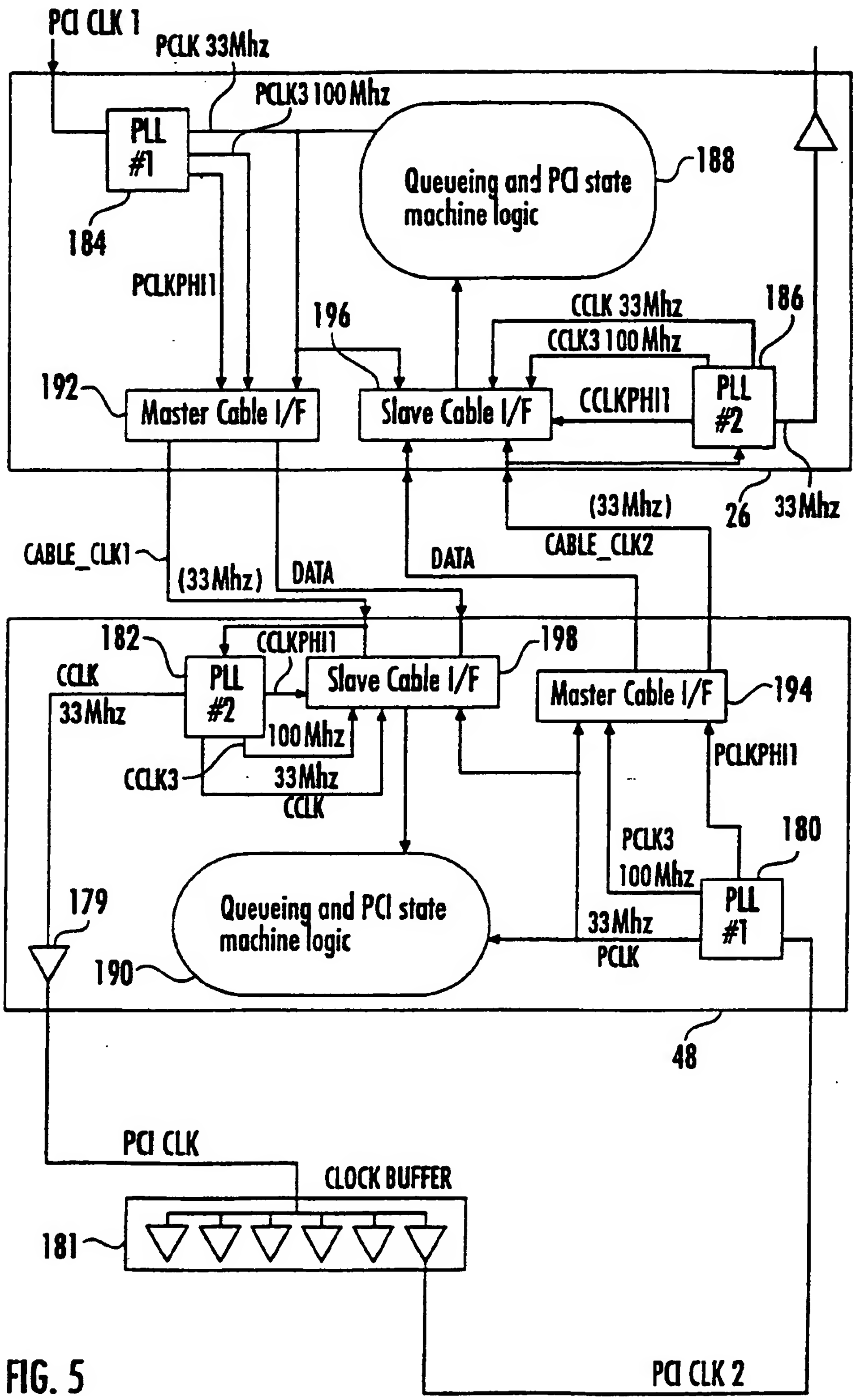

PCI CLK 1
PCLK 33Mhz
PCLK3 100 Mhz
PLL #1
184
Queueing and PCI state machine logic
188
PCLKPHI1
196
CCLK 33Mhz
186
CCLK3 100 Mhz
192
Master Cable I/F
Slave Cable I/F
CCLKPHI1
PLL #2
(33Mhz)
26
33Mhz
CABLE_CLK1
(33Mhz)
DATA
DATA
CABLE_CLK2
182
CCLKPHI1
198
CCLK 33Mhz
PLL #2
Slave Cable I/F
Master Cable I/F
194
100 Mhz
CCLK3
33Mhz CCLK
PCLKPHI1
PCLK3 100 Mhz
180
179
Queueing and PCI state machine logic
33Mhz PCLK
PLL #1
190
48
PCI CLK
CLOCK BUFFER
181
PCI CLK 2

FIG. 5

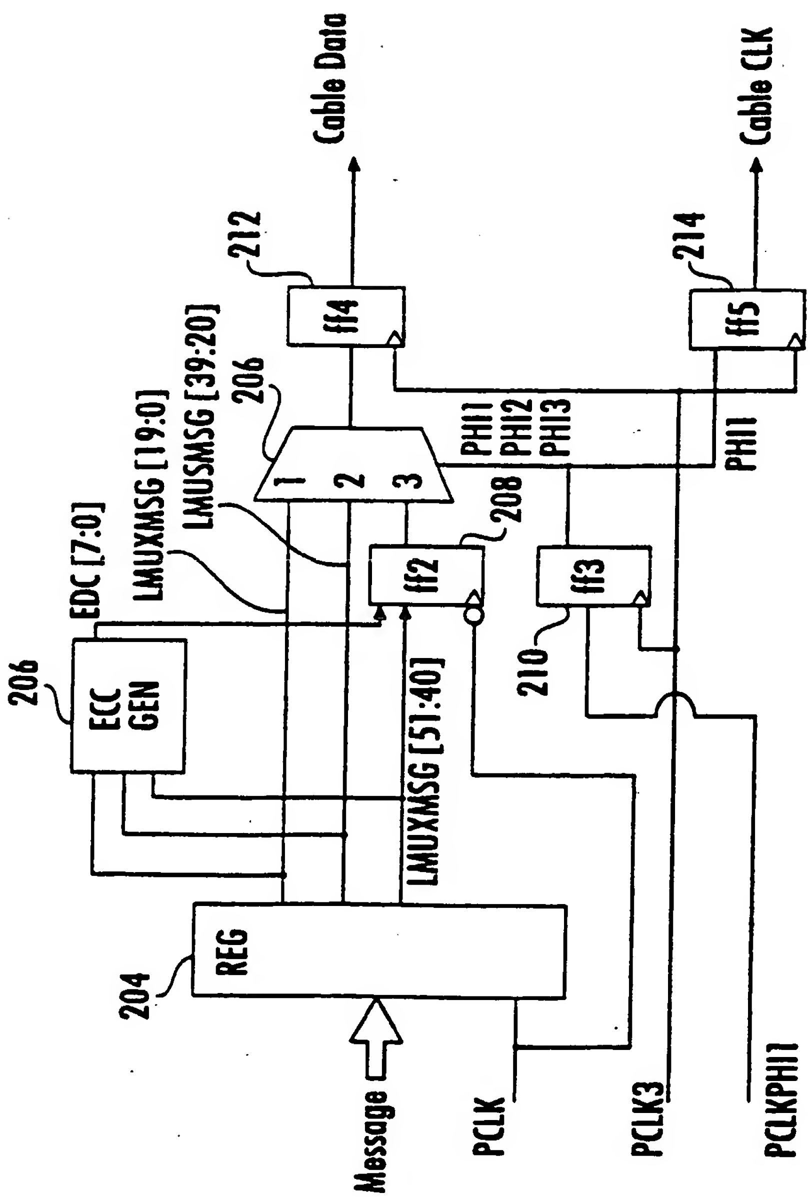

Cable Data
Cable CLK
212
214
ff4
ff5
206
LMUSMSG [39:20]
LMUXMSG [19:0]
EDC [7:0]
1
2
3
PHI1
PHI2
PHI3
PHI1
208
ff2
ff3
210
206
ECC
GEN
LMUXMSG [51:40]
204
REG
Message
PCLK
PCLK3
PCLKPHI1

FIG. 7

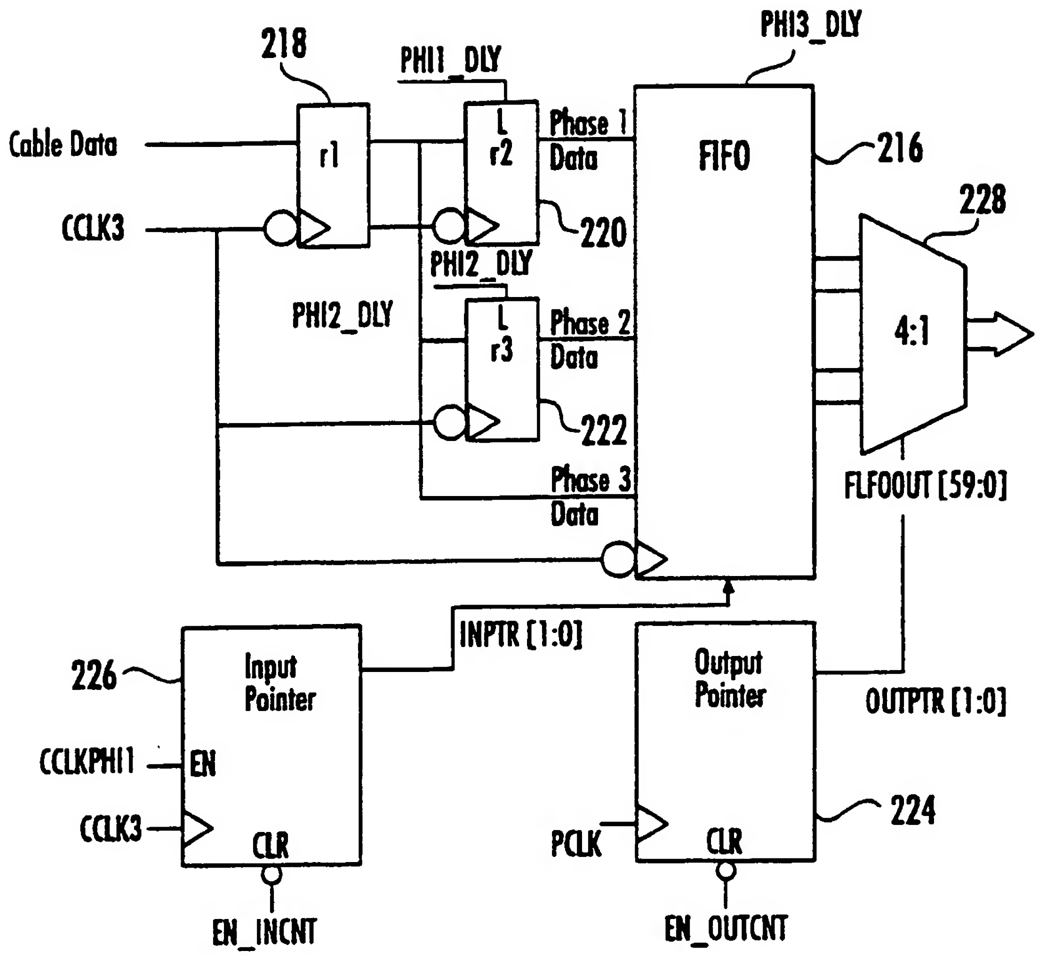

218
PHI1_DLY
PHI3_DLY
Cable Data
r1
L
r2
Phase 1
Data
FIFO
216
CCLK3
220
228
PHI2_DLY
PHI2_DLY
L
r3
Phase 2
Data
4:1
222
Phase 3
Data
FLFOOUT [59:0]
INPTR [1:0]
226
Input
Pointer
Output
Pointer
OUTPTR [1:0]
CCLKPHI1
EN
224
CCLK3
CLR
PCLK
CLR
EN_INCNT
EN_OUTCNT

FIG. 9

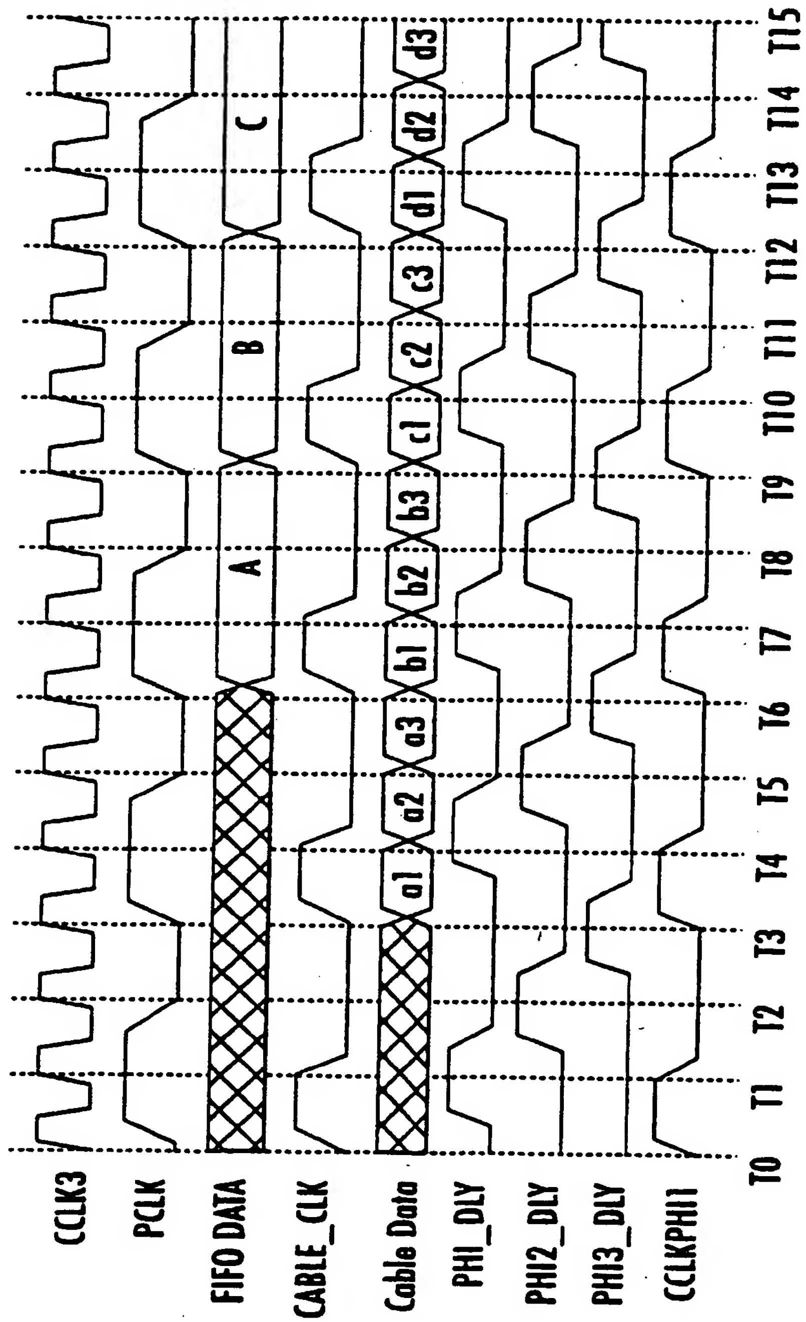

CCLK3
PCLK
FIFO DATA
CABLE_CLK
Cable Data
PHI_DLY
PHI2_DLY
PHI3_DLY
CCLKPHI1
A
B
C
a1
a2
a3
b1
b2
b3
c1
c2
c3
d1
d2
d3
T0
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
T13
T14
T15

FIG. 11

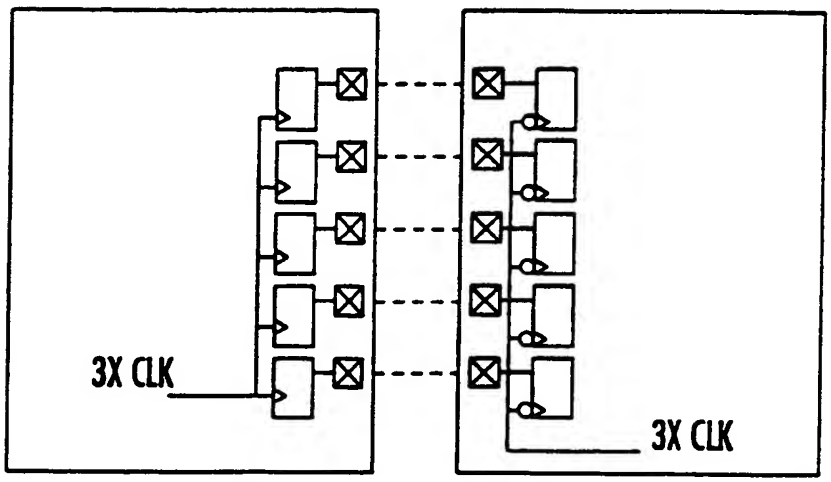


FIG. 13

Single Address Cycle		1st phase	2nd phase	subsequent phases
Delayed Read/Write Request	cbuff <3>	buff#	X	NA
	cbuff <2>	buff#	X	NA
	cbuff<1>	buff#	X	NA
	cbuff<0>	buff#	parity[3]	NA
	cbe<3:0>	PCI cmd	BE<>[1]	NA
	cad<>	addr	data<>[3]	NA
Posted Memory Write	cbuff <3>	X	X	X
	cbuff <2>	X	data ready	data ready
	cbuff<1>	X	parity error	parity error
	cbuff<0>	X	parity	parity
	cbe<3:0>	PCI cmd	BE<>	BE<>
	cad<>	addr	data	data
Delayed Read/Write Completion	cbuff <3>	buff#	end of completion	end of completion
	cbuff <2>	buff#	data ready	data ready
	cbuff<1>	buff#	parity error	parity error
	cbuff<0>	buff#	parity	parity
	cbe<3:0>	DRC	status	status
	cad<>	X	data	data
Stream Connect	cbuff <3>	buff#	X	X
	cbuff <2>	buff#	data ready	data ready
	cbuff<1>	buff#	X	X
	cbuff<0>	buff#	X	X
	cbe<3:0>	strm conn	X	X
	cad<>	X	X	X

FIG. 15A

Parameter	Value
Impedance (Differential)	108 +/- 5 Ohms
Impedance (Single-ended)	67 +/- 5 Ohms
Propagation Delay	1.54 ns/ft min, 1.58 ns/ft max
Delay Skew	0.025 ns/ft max
Attenuation (Differential)	0.08 db/ft max @ 50 MHz
Length	12'
DC Resistance	0.070 Ohms/ft max

FIG. 16

Data Bits

FIFO OUT [0:59]

CHK BIT [ ]	0123 4567	11 1111 8901 2345	1111 2222 6789 0123	2222 2233 4567 8901	3333 3333 2345 6789	4444 4444 0123 4567	4455 5555 8901 2345	5555 6789
0	0000 0001	1100 1100	1000 0011	1010 0000	1111 0011	0000 0110	1101 1000	0000
1	1100 1100	0110 1011	1000 0001	0001 0000	0001 0110	1001 0011	1010 0100	0000
2	0011 0110	0100 0000	1000 0100	1001 1100	1101 0101	0110 1000	0110 0010	0000
3	0000 1000	0001 0000	1000 1000	0100 0111	0011 1111	1111 0101	0001 0001	0000
4	1011 0001	0010 0001	0110 1110	0110 0010	0000 0000	1100 0000	1111 0000	1000
5	0000 0111	0001 0111	0111 0100	0100 1010	1001 1000	0011 0010	0000 0000	0100
6	1110 0000	1001 1001	1101 0000	0010 1011	0110 0000	0000 1001	0000 0000	0010
7	0101 1010	1010 0111	0011 1011	1001 0111	0000 1000	0000 1000	0000 0000	0001

FIG. 18

60 DB59,19	80 DB59	A0 DB 31,11	C0 UNCER	E0 DB19
61 UNCER	81 UNCER	A1 DB13	C1 DB08	E1 UNCER
62 UNCER	82 DB40,20	A2 DB14	C2 DB01	E2 UNCER
63 UNCER	83 DB23	A3 DB53,13	C3 UNCER	E3 UNCER
64 DB28	84 DB50,10	A4 DB06	C4 DB44	E4 DB33,13
65 UNCER	85 DB24	A5 DB28,08	C5 DB22,02	E5 UNCER
66 DB32,12	86 DB27	A6 DB54,14	C6 UNCER	E6 DB53,33,13
67 DB52,32,12	87 DB46,06	A7 UNCER	C7 UNCER	E7 DB36,16
68 DB11	88 UNCER	A8 DB36	C8 DB31	E8 UNCER
69 DB30,10	89 DB45	A9 DB43,23	C9 UNCER	E9 DB42,22,02
6A DB30,10	8A DB04	AA UNCER	CA DB20,00	EA DB41,21,0
6B UNCER	8B DB29,09	AB UNCER	CB DB44,24,04	EB DB34,14
6C UNCER	8C DB29	AC UNCER	CC DB47,27	EC UNCER
6D DB59,39,19	8D DB59,39	AD UNCER	CD UNCER	ED DB39,19
6E UNCER	8E UNCER	AE UNCER	CE UNCER	EE DB50,30
6F UNCER	8F UNCER	AF DB45,05	CF UNCER	EF DB54,34,14
70 DB17	90 UNCER	B0 DB18	D0 DB40,20,00	F0 DB 58,18
71 DB51,11	91 DB22	B1 DB45,25	D1 DB51,31	F1 UNCER
72 DB46,26	92 DB10	B2 UNCER	D2 48,08	F2 DB15
73 UNCER	93 UNCER	B3 UNCER	D3 UNCER	F3 UNCER
74 UNCER	94 DB03	B4 UNCER	D4 UNCER	F4 UNCER
75 UNCER	95 UNCER	B5 UNCER	D5 DB55,35,15	F5 DB26,06
76 UNCER	96 UNCER	B6 DB48,28,08	D6 DB46,26,06	F6 DB21,01
77 DB48,28	97 DB45,25,05	B7 DB27,07	D7 UNCER	F7 DB56,36,16
78 DB42,02	98 DB20	B8 DB56,36	D8 UNCER	F8 DB30
79 UNCER	99 DB49,29	B9 DB51,31,11	D9 UNCER	F9 UNCER
7A UNCER	9A UNCER	BA UNCER	DA UNCER	FA DB55,15
7B DB47,07	9B UNCER	BB DB38,18	DB UNCER	FB DB58,38,18
7C DB50,30,10	9C UNCER	BC UNCER	DC UNCER	FC UNCER
7D UNCER	9D UNCER	BD DB42,22	DD DB35,15	FD DB47,27,0
7E DB37,17	9E DB49,29,09	BE DB43,03	DE DB41,01	FE UNCER
7F UNCER	9F UNCER	BF UNCER	DF UNCER	FF UNCER

FIG. 19B

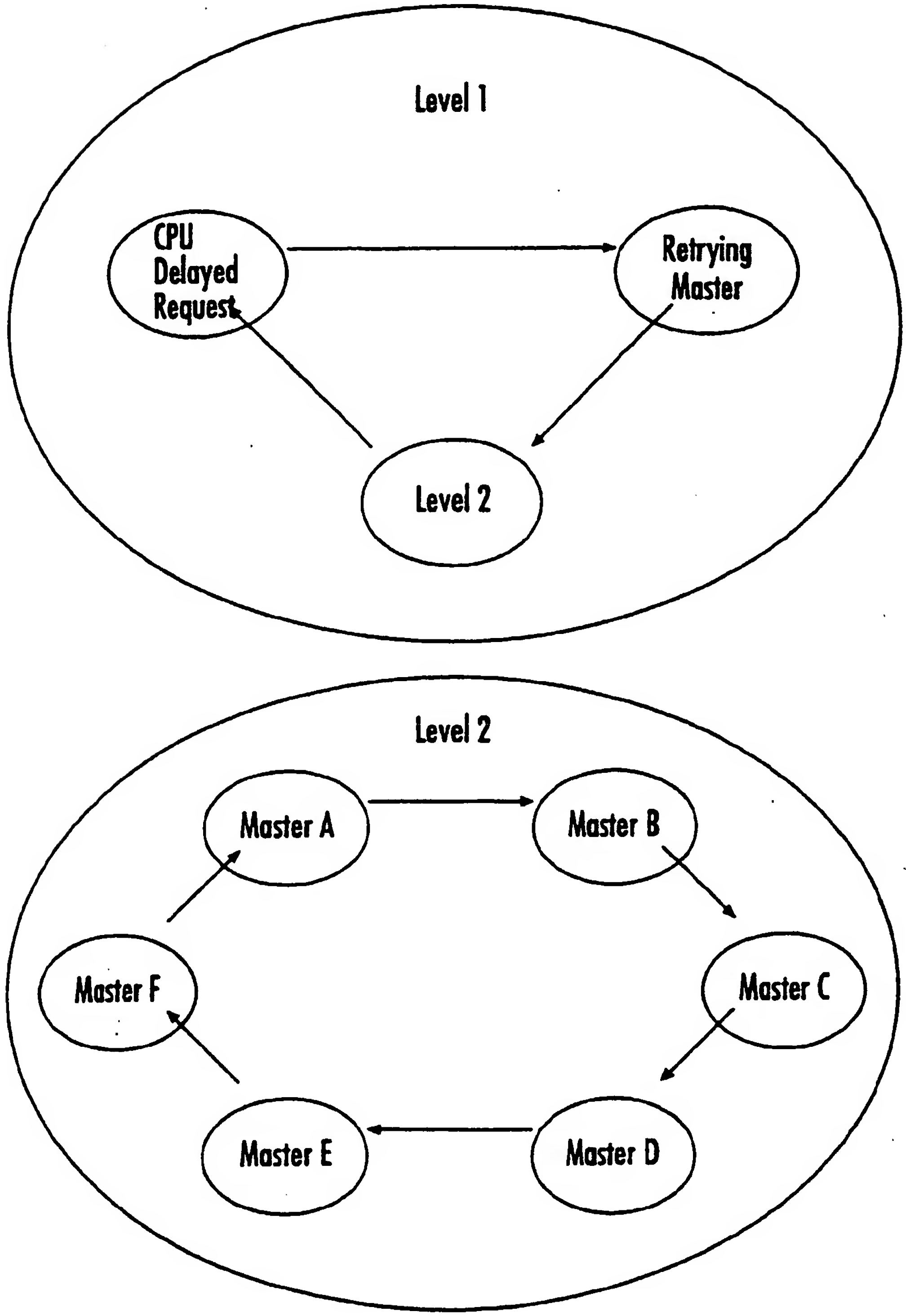

Level 1
CPU Delayed Request
Retrying Master
Level 2
Level 2
Master A
Master B
Master C
Master D
Master E
Master F

FIG 20B

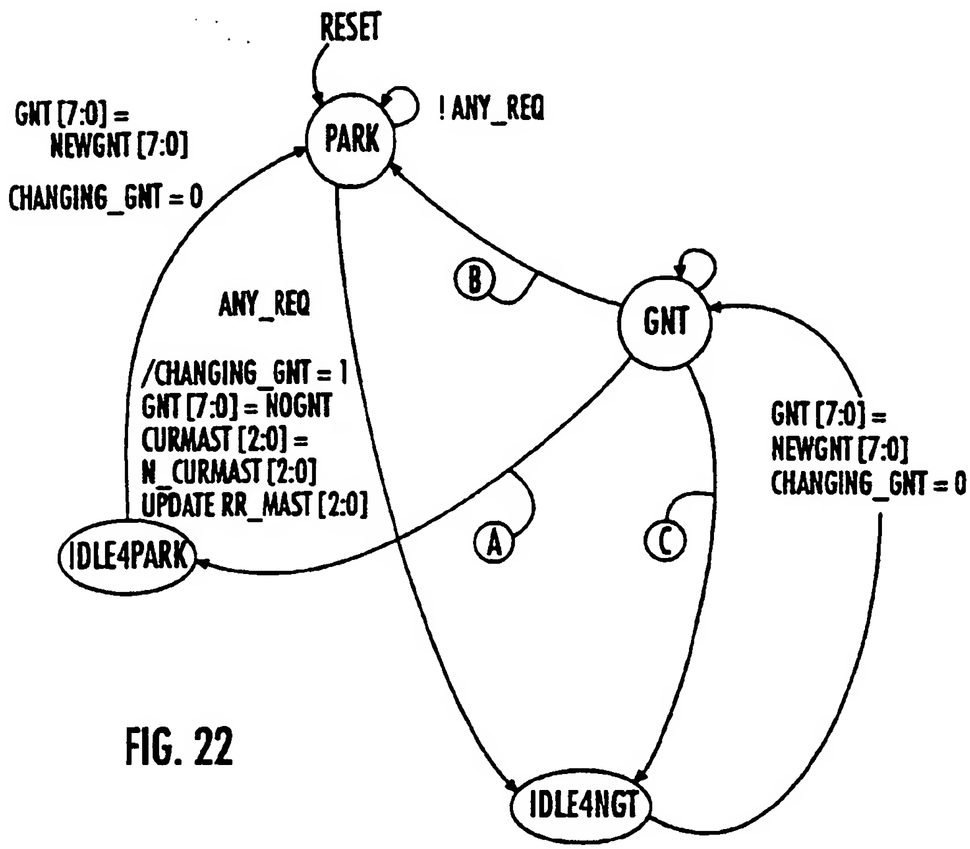


FIG. 22

(A) OPEN_WINDOW & !ANY_REQ & BUS_IDLE & (N_CURMAST! = CURMAST) / CHANGING_GNT = 1
GNT [7:0] = NOGNT
CURMAST [2:0] = N_CURMAST [2:0]
RR_MAST [2:0]
=N_RR_MAST [2:0]
L1STATE [1:0] =
N_L1STATE [1:0]

(B) OPEN_WINDOW & !ANY_REQ & BUS_IDLE & (N_CURMAST = CURMAST)
/L1STATE [1:0] = N_L1STATE [1:0]

(C) OPEN_WINDOW & (N_CURMAST! = CURMAST) / CHANGING_GNT = 1
GNT [7:0] = NOGNT
CURMAST [2:0] = N_CURMAST [2:0]
UPDATE RR_MAST [2:0]
L1STATE [1:0] = N_L1STATE [1:0]

CURMAST [2:0]	NEWGNT [7:0]
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

FIG. 24

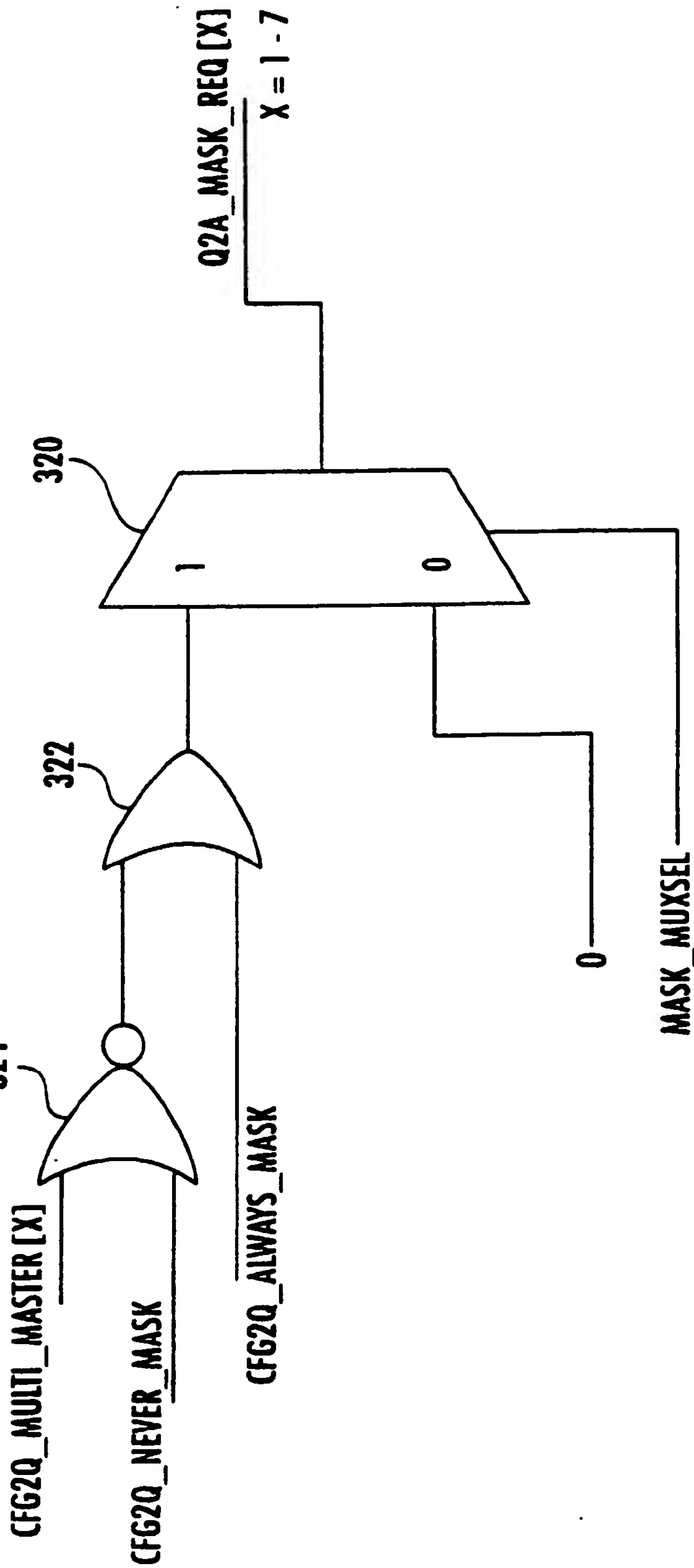

Q2A_MASK_REQ [X]
X = 1 - 7
320
1
0
322
324
CFG2Q_MULTI_MASTER [X]
CFG2Q_NEVER_MASK
CFG2Q_ALWAYS_MASK
0
MASK_MUXSEL

FIG. 26A

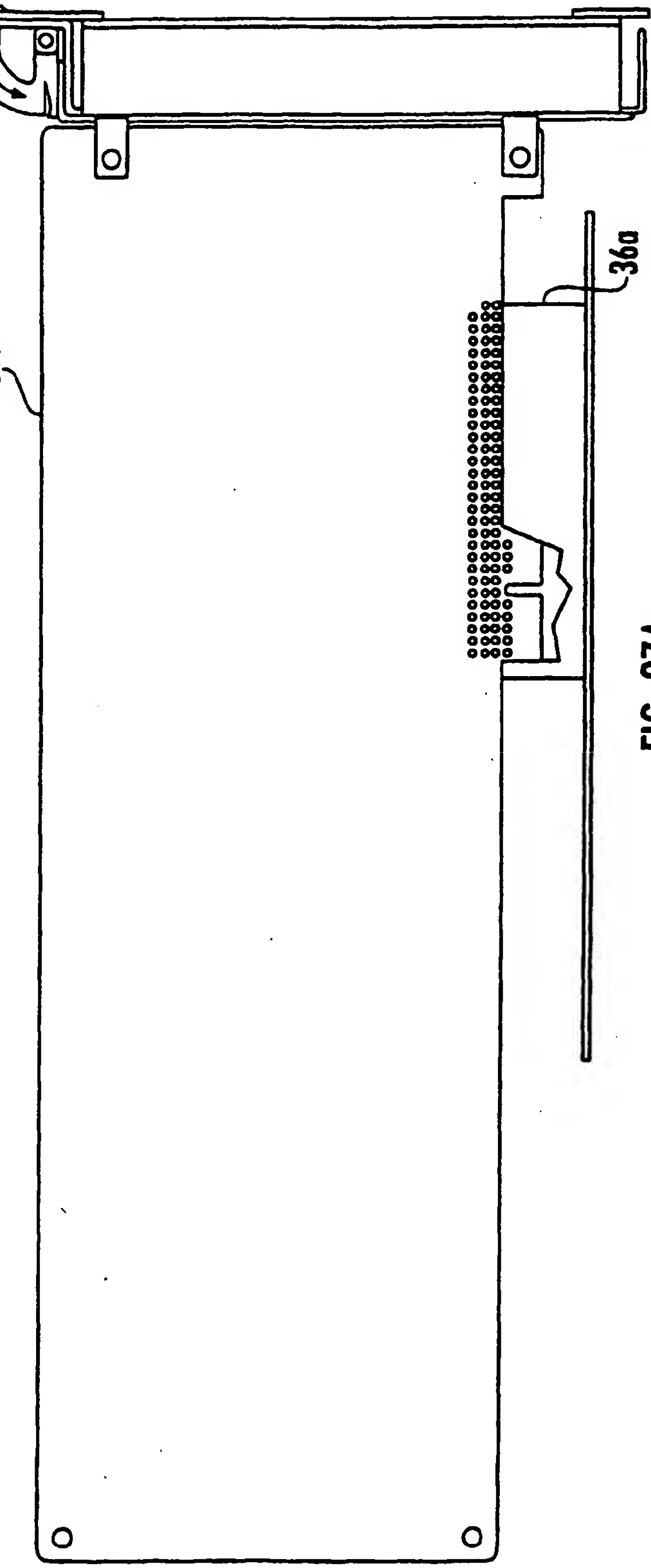


FIG. 27A

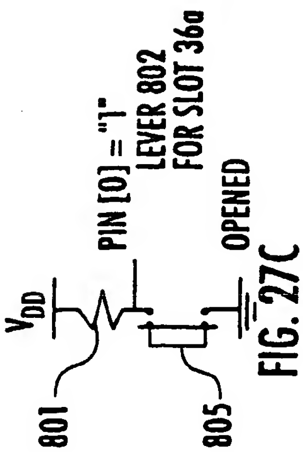


FIG. 27C

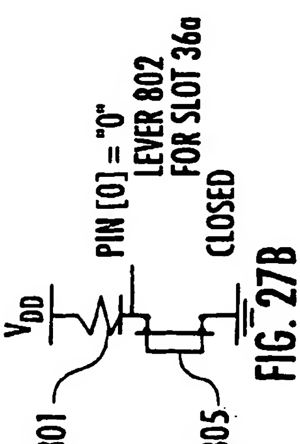


FIG. 27B

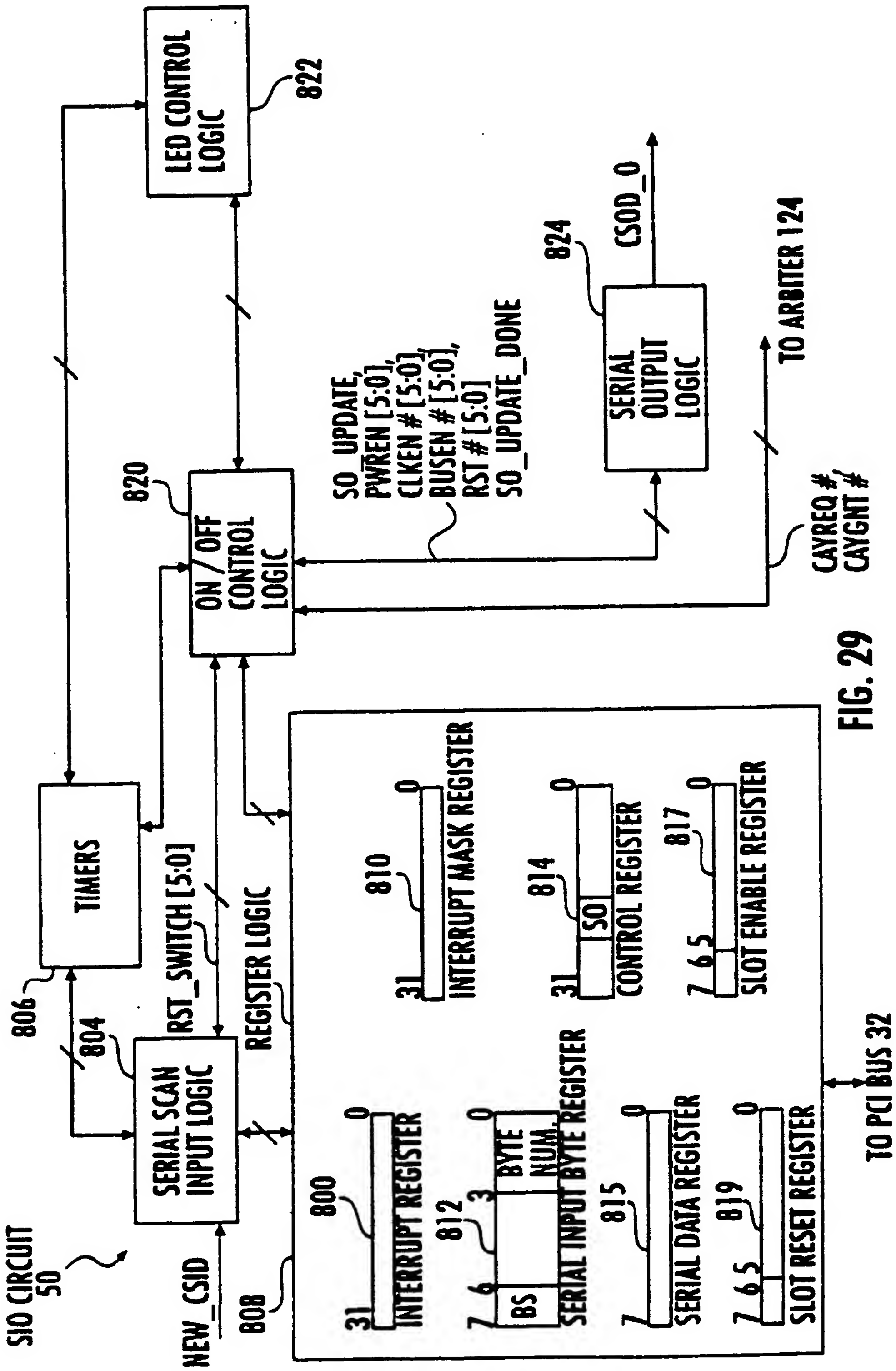

SIO CIRCUIT 50
NEW_CSID
808
SERIAL SCAN INPUT LOGIC
804
806
TIMERS
RST_SWITCH [5:0]
REGISTER LOGIC
820
ON / OFF CONTROL LOGIC
LED CONTROL LOGIC
822
SO_UPDATE,
PWREN [5:0],
CLKEN # [5:0],
BUSEN # [5:0],
RST # [5:0]
SO_UPDATE_DONE
824
SERIAL OUTPUT LOGIC
CSOD_O
TO ARBITER 124
CAYREQ #,
CAYGNT #
800
31
0
INTERRUPT REGISTER
812
7 6 3 0
BS
BYTE NUM
SERIAL INPUT BYTE REGISTER
815
7 0
SERIAL DATA REGISTER
819
7 6 5 0
SLOT RESET REGISTER
810
31 0
INTERRUPT MASK REGISTER
814
31 0
SO
CONTROL REGISTER
817
7 6 5 0
SLOT ENABLE REGISTER
TO PCI BUS 32

FIG. 29

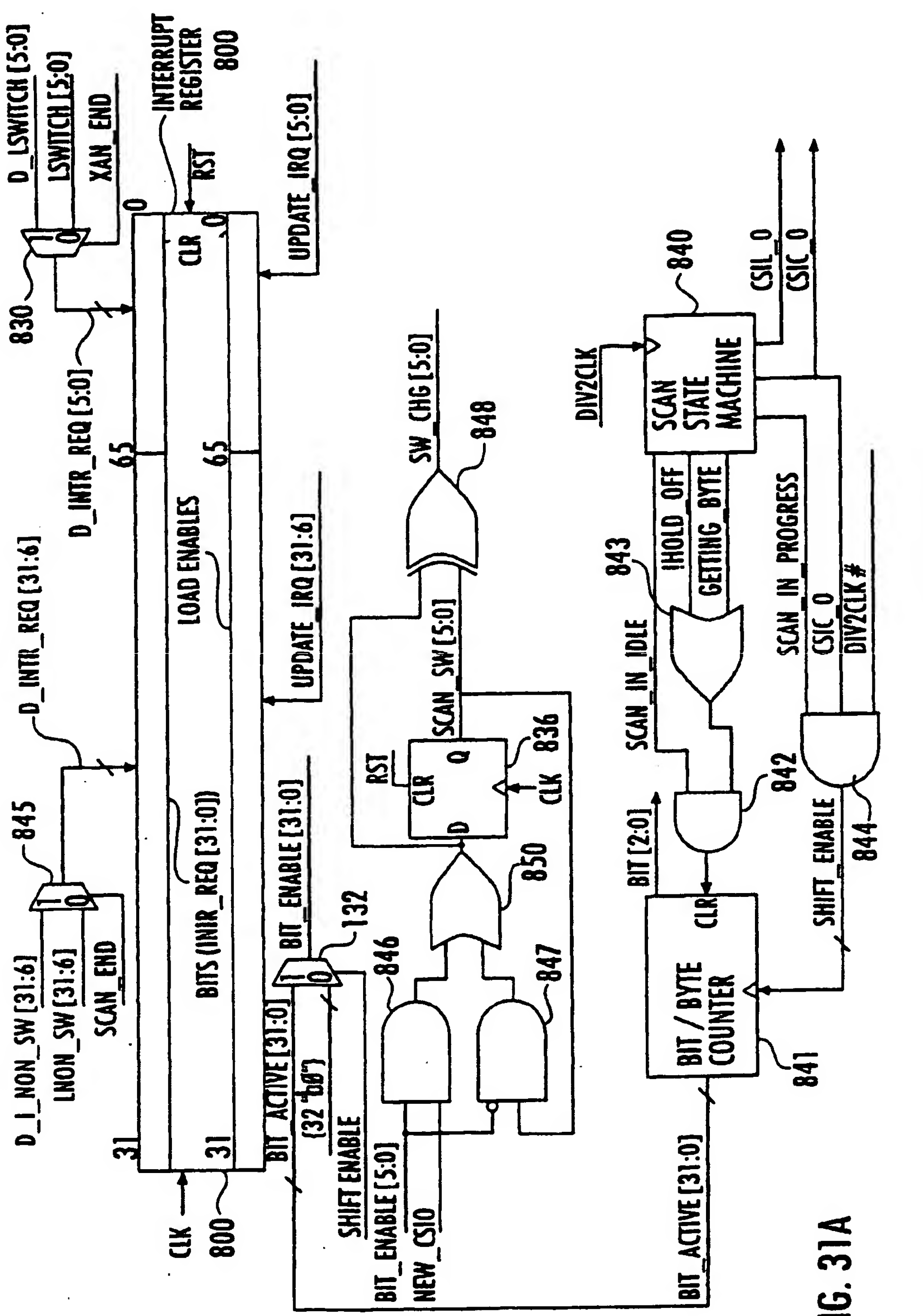
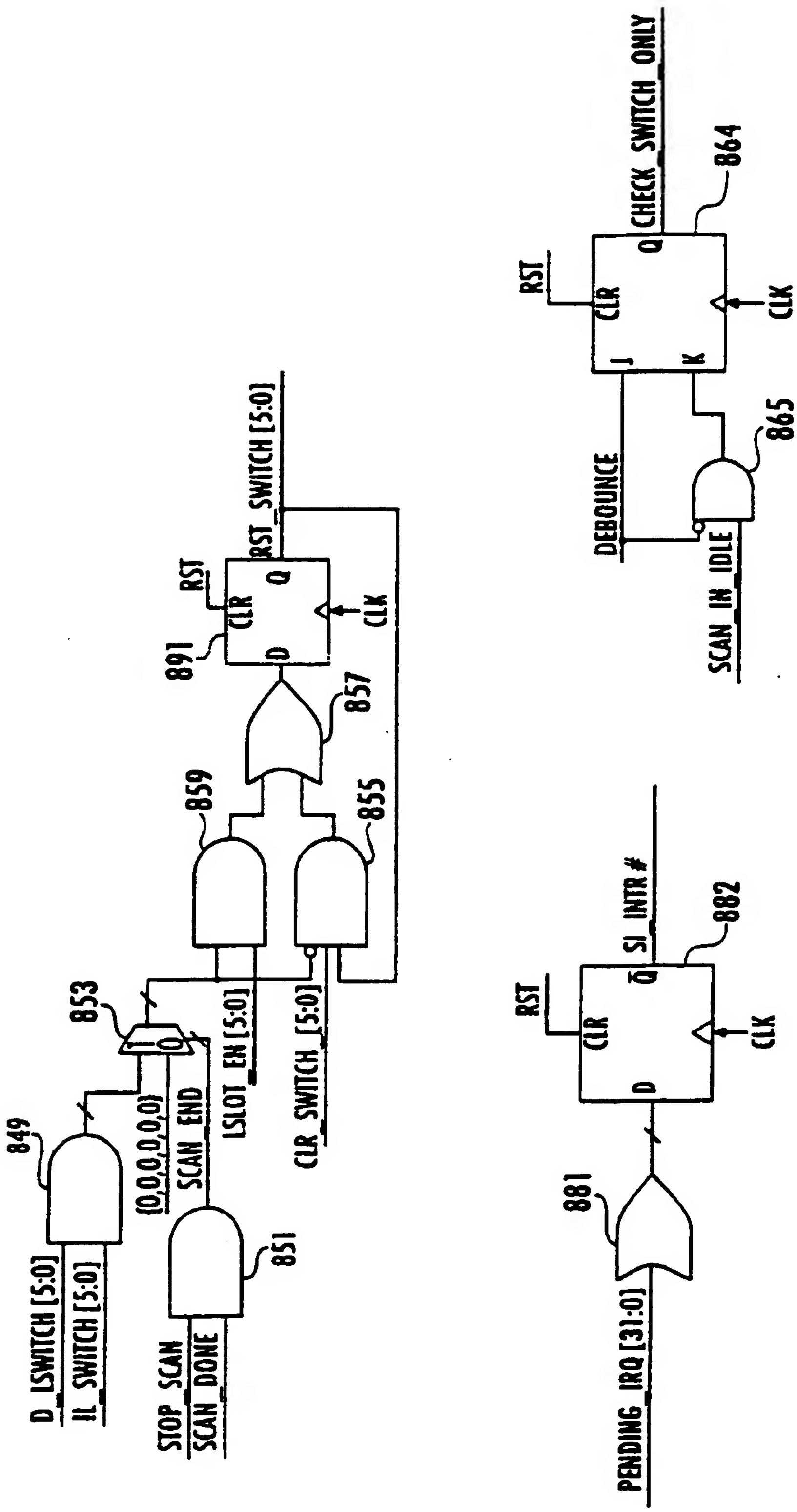

D_LSWITCH [5:0]
LSWITCH [5:0]
XAN_END
INTERRUPT REGISTER 800
RST
CLR
830
D_INTR_REQ [5:0]
UPDATE_IRQ [5:0]
65
65
0
0
1
0
D_INTR_REQ [31:6]
LOAD ENABLES
UPDATE_IRQ [31:6]
845
BITS (INTR_REQ [31:0])
D_I_NON_SW [31:6]
LNON_SW [31:6]
SCAN_END
31
31
CLK
800
BIT_ENABLE [31:0]
132
BIT_ACTIVE [31:0]
{32 'b0'}
SHIFT ENABLE
BIT_ENABLE [5:0]
NEW_CSIO
846
847
850
D
CLR
Q
CLK
836
SCAN_SW [5:0]
848
SW_CHG [5:0]
DIV2CLK
SCAN STATE MACHINE
840
CSIL_0
CSIC_0
!HOLD_OFF
GETTING_BYTE
843
SCAN_IN_IDLE
SCAN_IN_PROGRESS
CSIC_0
DIV2CLK #
842
844
BIT [2:0]
SHIFT_ENABLE
BIT / BYTE COUNTER
CLR
841
BIT_ACTIVE [31:0]

FIG. 31A



D_LSWITCH [5:0]
IL_SWITCH [5:0]
849
{0,0,0,0,0,0}
STOP_SCAN
SCAN_DONE
851
SCAN_END
853
1
0
LSLOT_EN [5:0]
CLR_SWITCH [5:0]
859
855
857
891
D
CLR
RST
Q
CLK
RST_SWITCH [5:0]
RST
CLR
J
K
Q
CLK
CHECK_SWITCH_ONLY
864
DEBOUNCE
SCAN_IN_IDLE
865
PENDING_IRQ [31:0]
881
D
CLR
RST
Q̄
CLK
SI_INTR#
882

FIG. 31B2

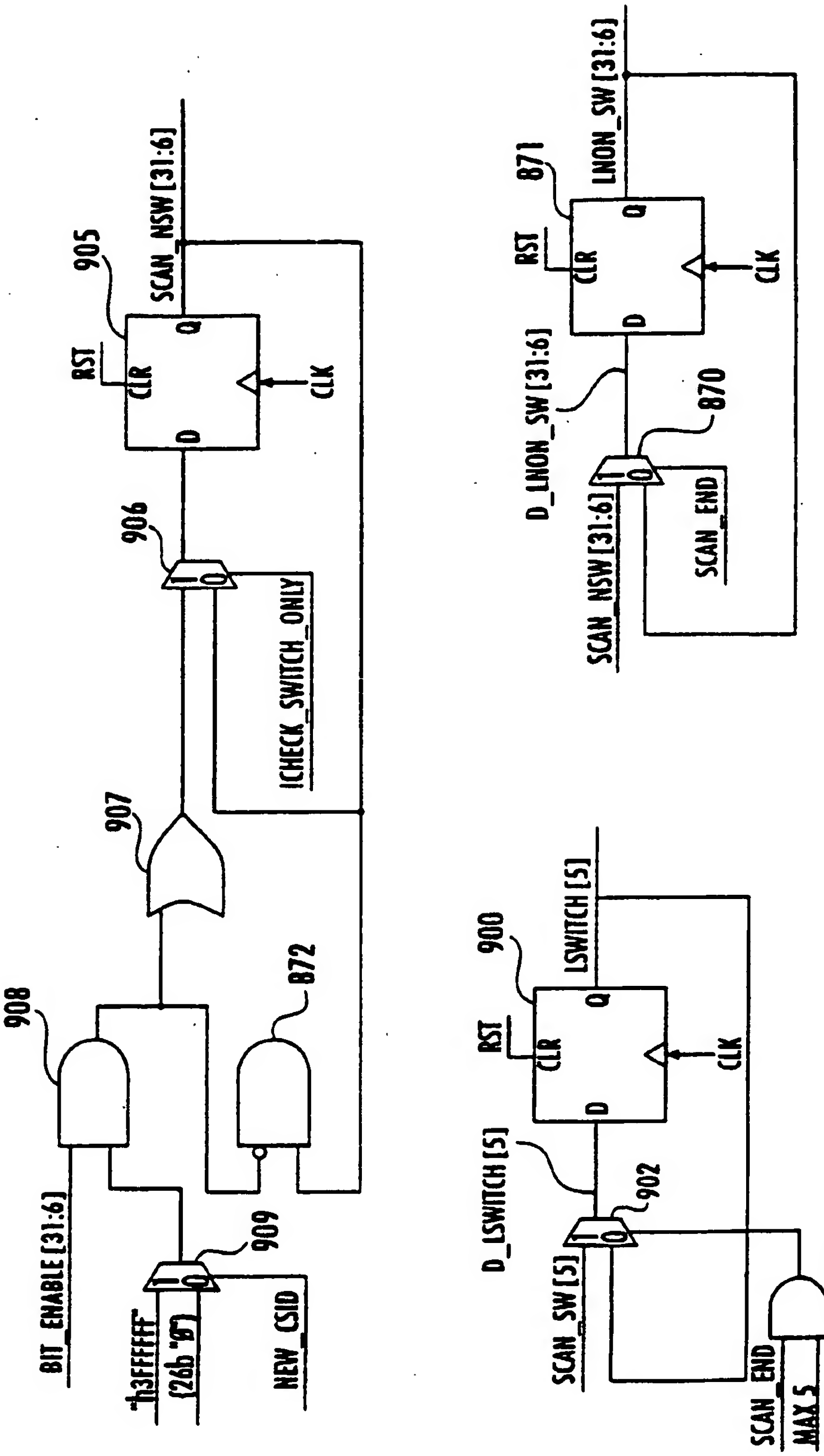
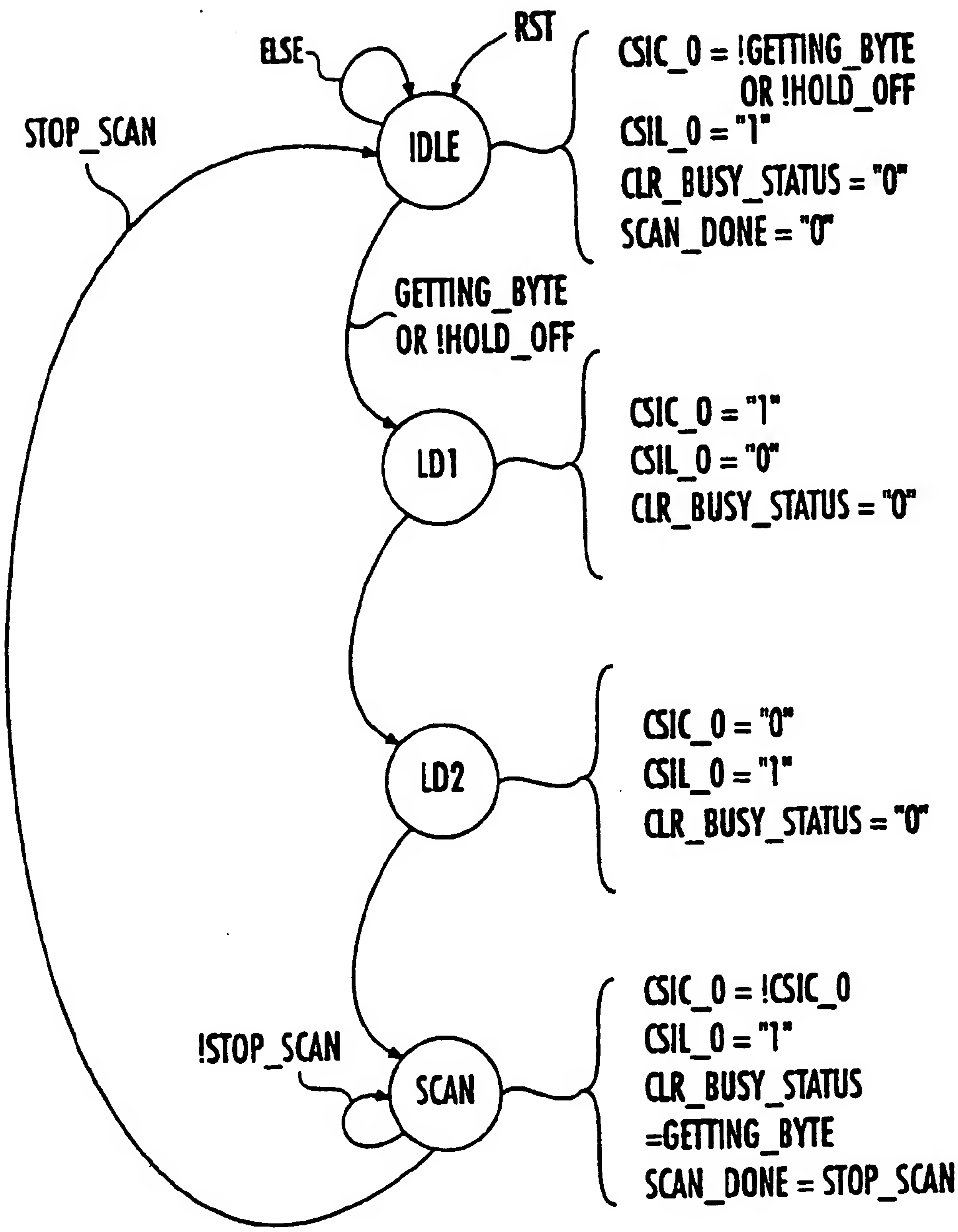

SCAN_NSW [31:6]
905
RST
CLR
Q
D
CLK
906
1
0
ICHECK_SWITCH_ONLY
907
908
872
BIT_ENABLE [31:6]
'h3FFFFFF
{26b'0}
909
NEW_CSID
LNON_SW [31:6]
871
RST
CLR
Q
D
CLK
D_LNON_SW [31:6]
870
1
0
SCAN_END
SCAN_NSW [31:6]
LSWITCH [5]
900
RST
CLR
Q
D
CLK
D_LSWITCH [5]
902
1
0
SCAN_SW [5]
903
SCAN_END
MAX_5

FIG. 31D



STOP_SCAN = (BYTE_PTR_EQUAL_CNT & GETTING_BYTE) OR ((BYTE [1] & BIT [0] & CHECK_SWITCH_ONLY) OR (BYTE [4] & BIT [0] & !CHECK_SWITCH_ONLY)) & !GETTING_BYTE

FIG. 32A

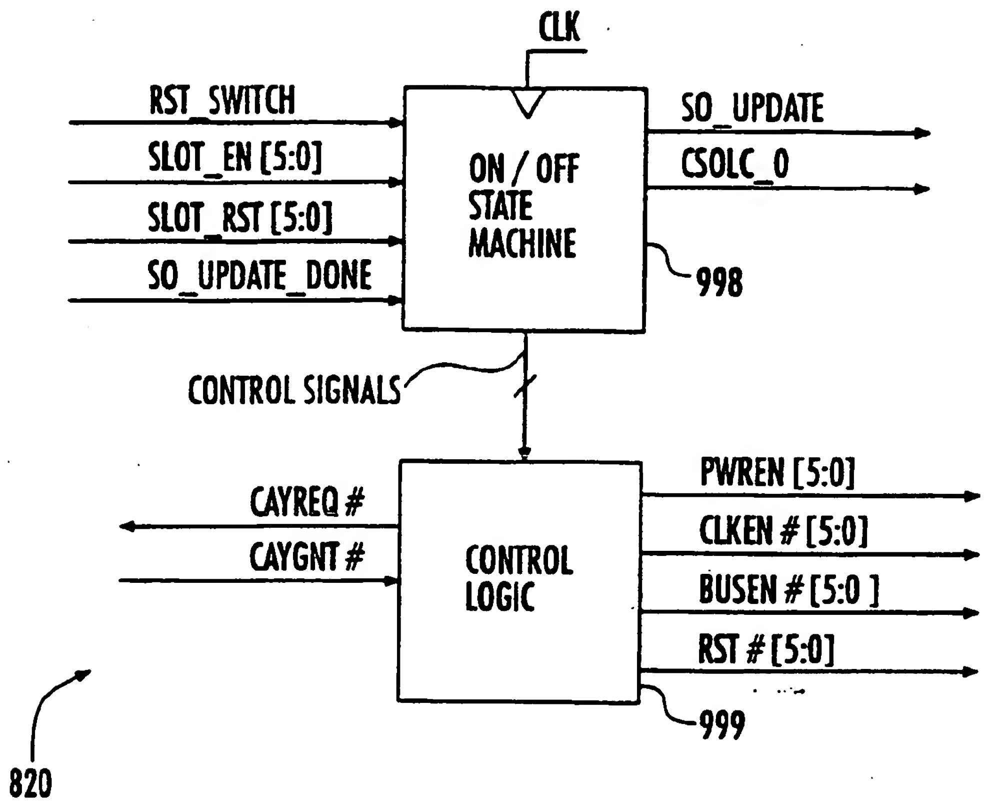

CLK
RST_SWITCH
SLOT_EN [5:0]
SLOT_RST [5:0]
SO_UPDATE_DONE
ON / OFF
STATE
MACHINE
SO_UPDATE
CSOLC_0
998
CONTROL SIGNALS
CAYREQ #
CAYGNT #
CONTROL
LOGIC
PWREN [5:0]
CLKEN # [5:0]
BUSEN # [5:0]
RST # [5:0]
999
820

FIG. 33A

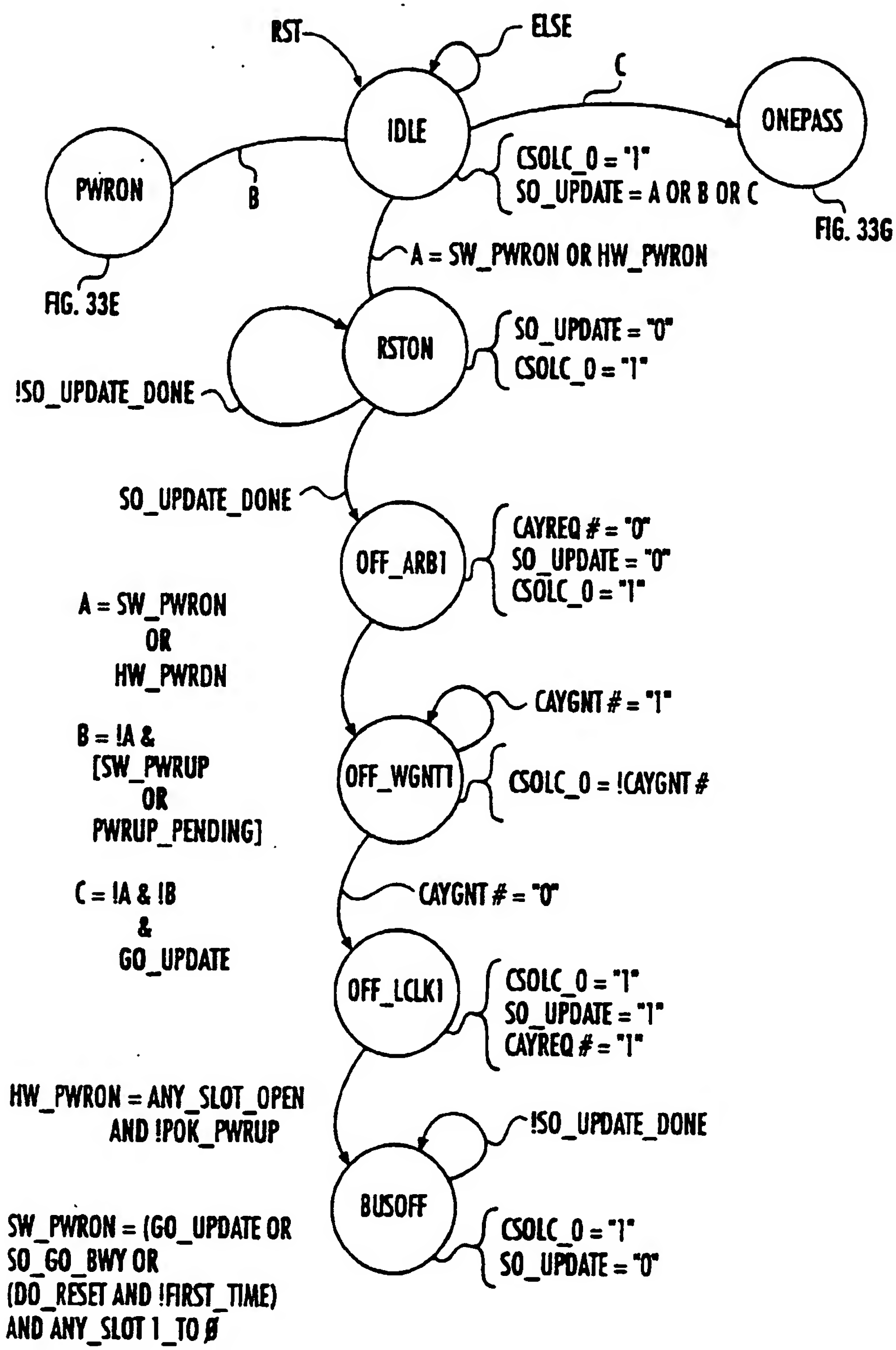

RST
ELSE
C
IDLE
ONEPASS
PWRON
B
CSOLC_0 = "1"
SO_UPDATE = A OR B OR C
FIG. 33G
A = SW_PWRON OR HW_PWRON
FIG. 33E
RSTON
SO_UPDATE = "0"
CSOLC_0 = "1"
!SO_UPDATE_DONE
SO_UPDATE_DONE
OFF_ARB1
CAYREQ # = "0"
SO_UPDATE = "0"
CSOLC_0 = "1"
A = SW_PWRON
OR
HW_PWRDN
CAYGNT # = "1"
OFF_WGNT1
CSOLC_0 = !CAYGNT #
B = !A &
[SW_PWRUP
OR
PWRUP_PENDING]
CAYGNT # = "0"
C = !A & !B
&
GO_UPDATE
OFF_LCLK1
CSOLC_0 = "1"
SO_UPDATE = "1"
CAYREQ # = "1"
HW_PWRON = ANY_SLOT_OPEN
AND !POK_PWRUP
!SO_UPDATE_DONE
BUSOFF
SW_PWRON = (GO_UPDATE OR
SO_GO_BWY OR
(DO_RESET AND !FIRST_TIME)
AND ANY_SLOT 1_TO Ø
CSOLC_0 = "1"
SO_UPDATE = "0"

FIG. 33C

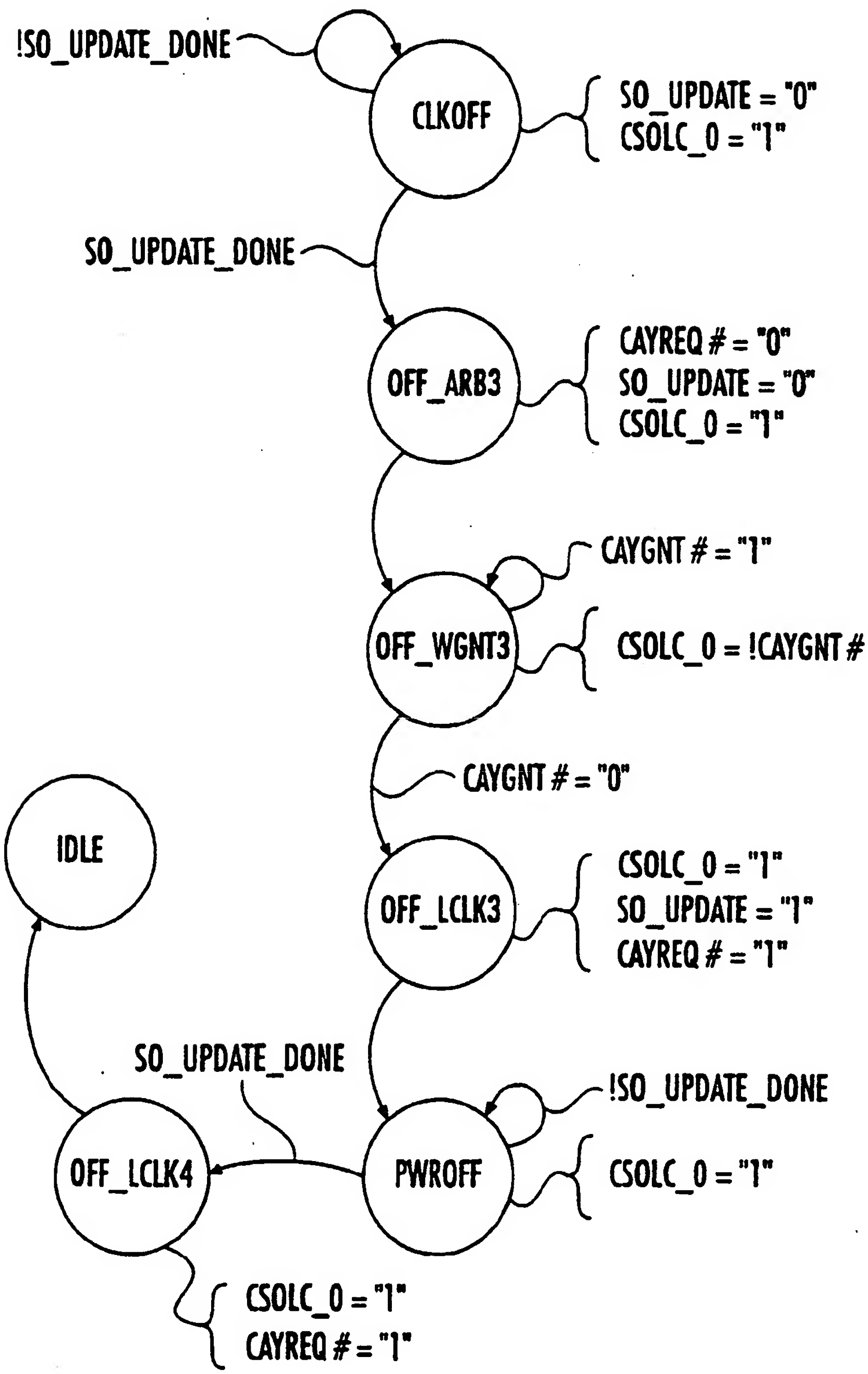

!SO_UPDATE_DONE
CLKOFF
SO_UPDATE = "0"
CSOLC_0 = "1"
SO_UPDATE_DONE
OFF_ARB3
CAYREQ # = "0"
SO_UPDATE = "0"
CSOLC_0 = "1"
CAYGNT # = "1"
OFF_WGNT3
CSOLC_0 = !CAYGNT #
CAYGNT # = "0"
IDLE
OFF_LCLK3
CSOLC_0 = "1"
SO_UPDATE = "1"
CAYREQ # = "1"
SO_UPDATE_DONE
!SO_UPDATE_DONE
OFF_LCLK4
PWROFF
CSOLC_0 = "1"
CSOLC_0 = "1"
CAYREQ # = "1"

FIG. 33E

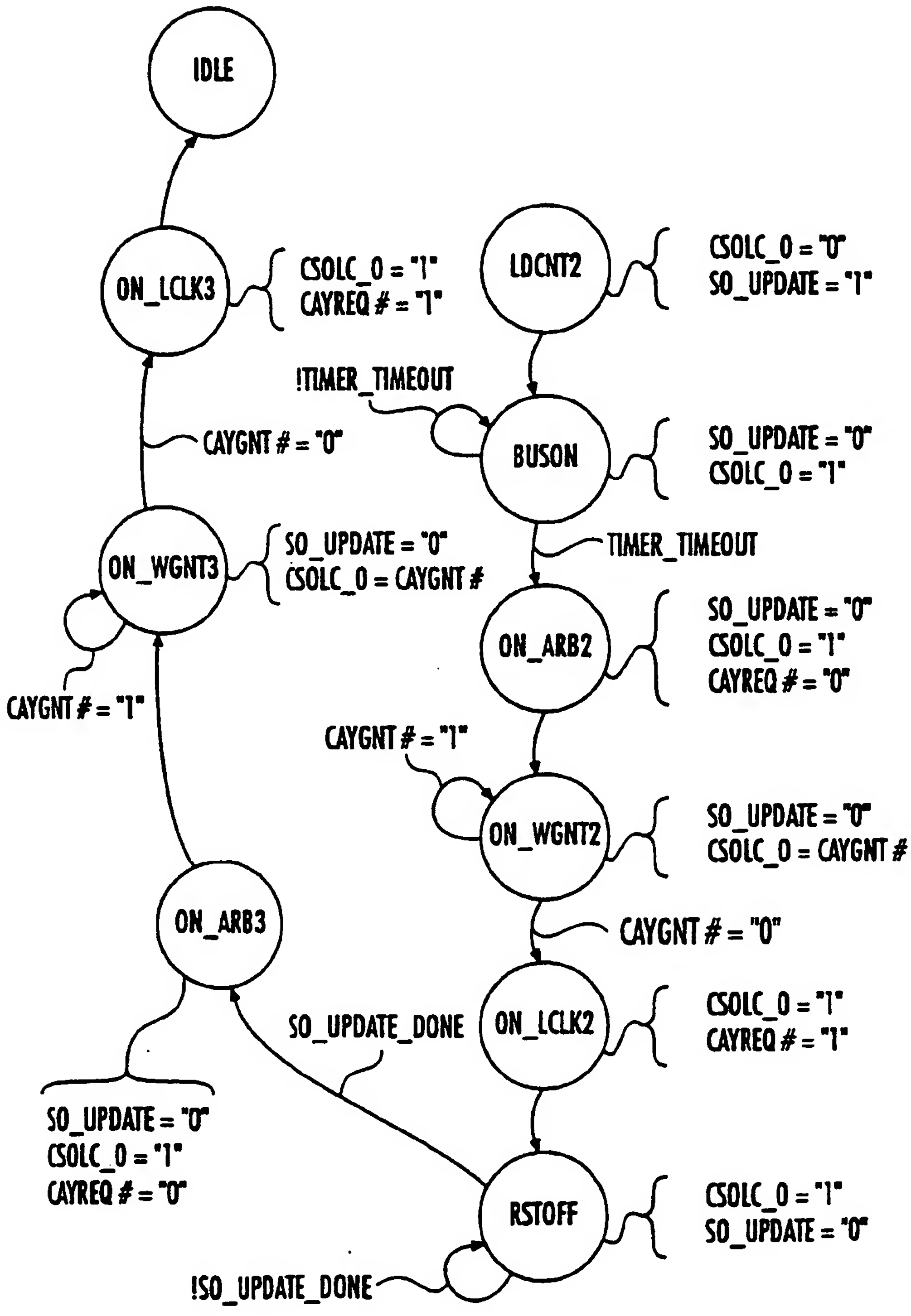

IDLE
ON_LCLK3
CSOLC_0 = "1"
CAYREQ # = "1"
LDCNT2
CSOLC_0 = "0"
SO_UPDATE = "1"
!TIMER_TIMEOUT
CAYGNT # = "0"
BUSON
SO_UPDATE = "0"
CSOLC_0 = "1"
ON_WGNT3
SO_UPDATE = "0"
CSOLC_0 = CAYGNT #
TIMER_TIMEOUT
ON_ARB2
SO_UPDATE = "0"
CSOLC_0 = "1"
CAYREQ # = "0"
CAYGNT # = "1"
CAYGNT # = "1"
ON_WGNT2
SO_UPDATE = "0"
CSOLC_0 = CAYGNT #
ON_ARB3
CAYGNT # = "0"
SO_UPDATE_DONE
ON_LCLK2
CSOLC_0 = "1"
CAYREQ # = "1"
SO_UPDATE = "0"
CSOLC_0 = "1"
CAYREQ # = "0"
RSTOFF
CSOLC_0 = "1"
SO_UPDATE = "0"
!SO_UPDATE_DONE

FIG. 33G

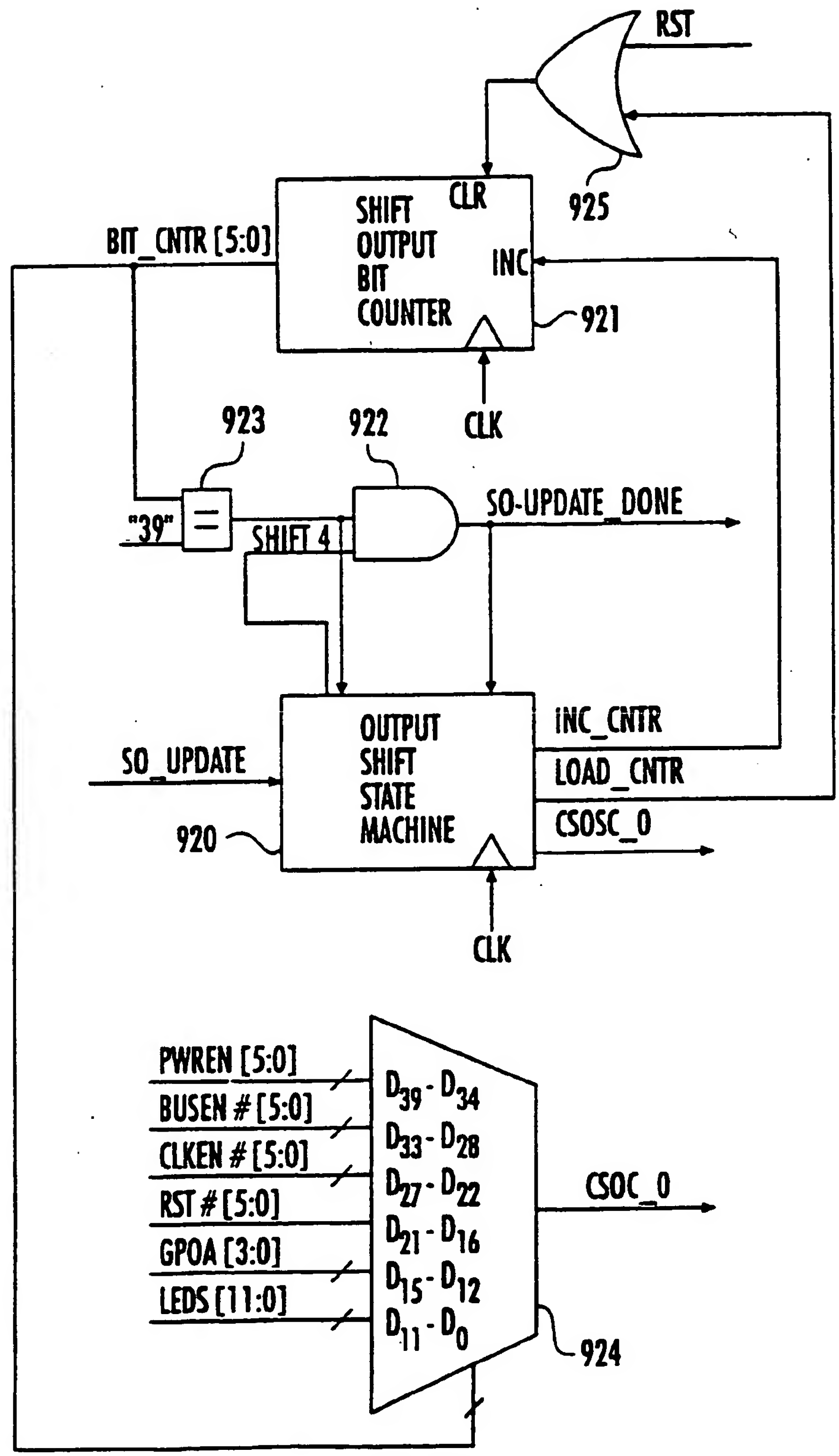

RST
925
SHIFT
OUTPUT
BIT
COUNTER
CLR
INC
921
BIT_CNTR [5:0]
CLK
923
922
"39"
=
SHIFT 4
SO-UPDATE_DONE
OUTPUT
SHIFT
STATE
MACHINE
INC_CNTR
LOAD_CNTR
CSOSC_0
SO_UPDATE
920
CLK
PWREN [5:0]
BUSEN # [5:0]
CLKEN # [5:0]
RST # [5:0]
GPOA [3:0]
LEDS [11:0]
D39 - D34
D33 - D28
D27 - D22
D21 - D16
D15 - D12
D11 - D0
CSOC_0
924

FIG. 34

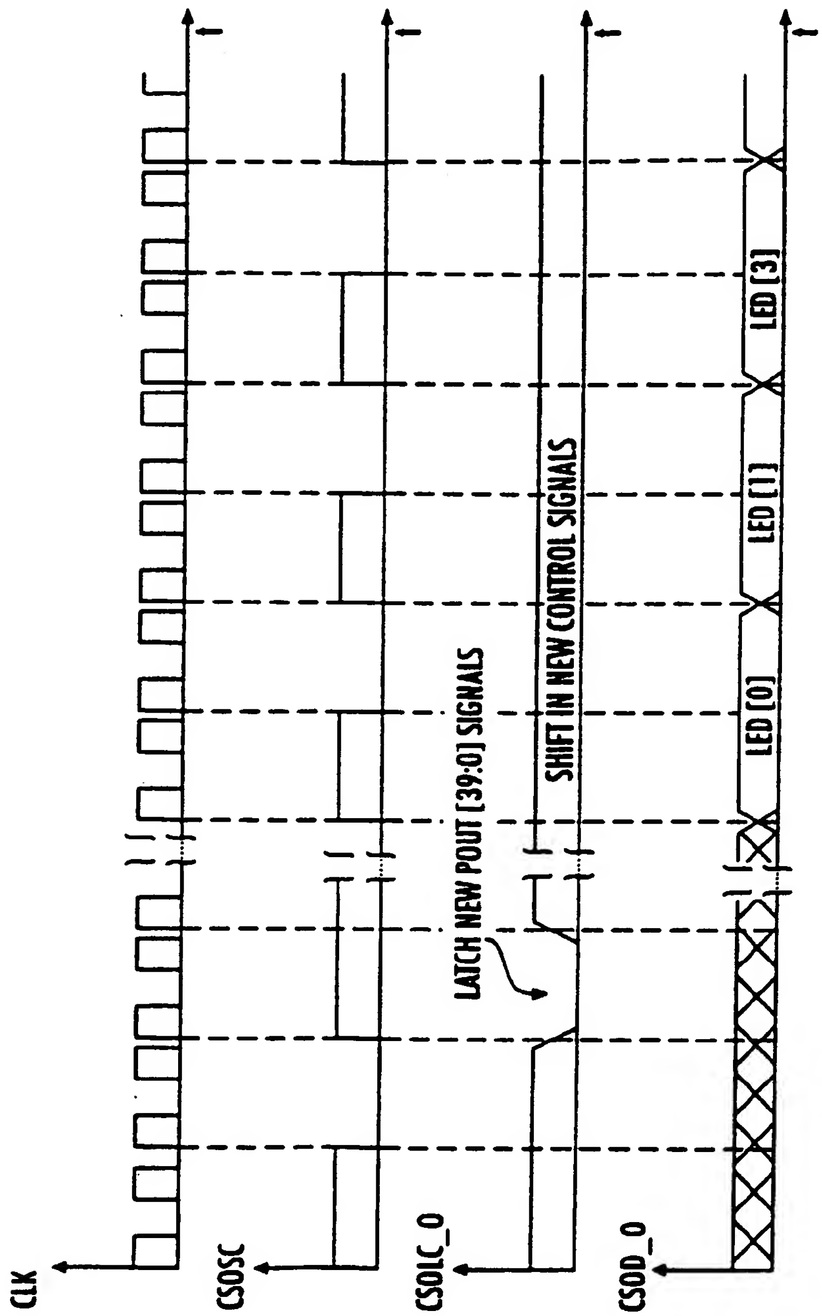

CLK
CSOSC
CSOLC_0
CSOD_0
LATCH NEW POUT [39:0] SIGNALS
SHIFT IN NEW CONTROL SIGNALS
LED [0]
LED [1]
LED [3]

FIG. 35B

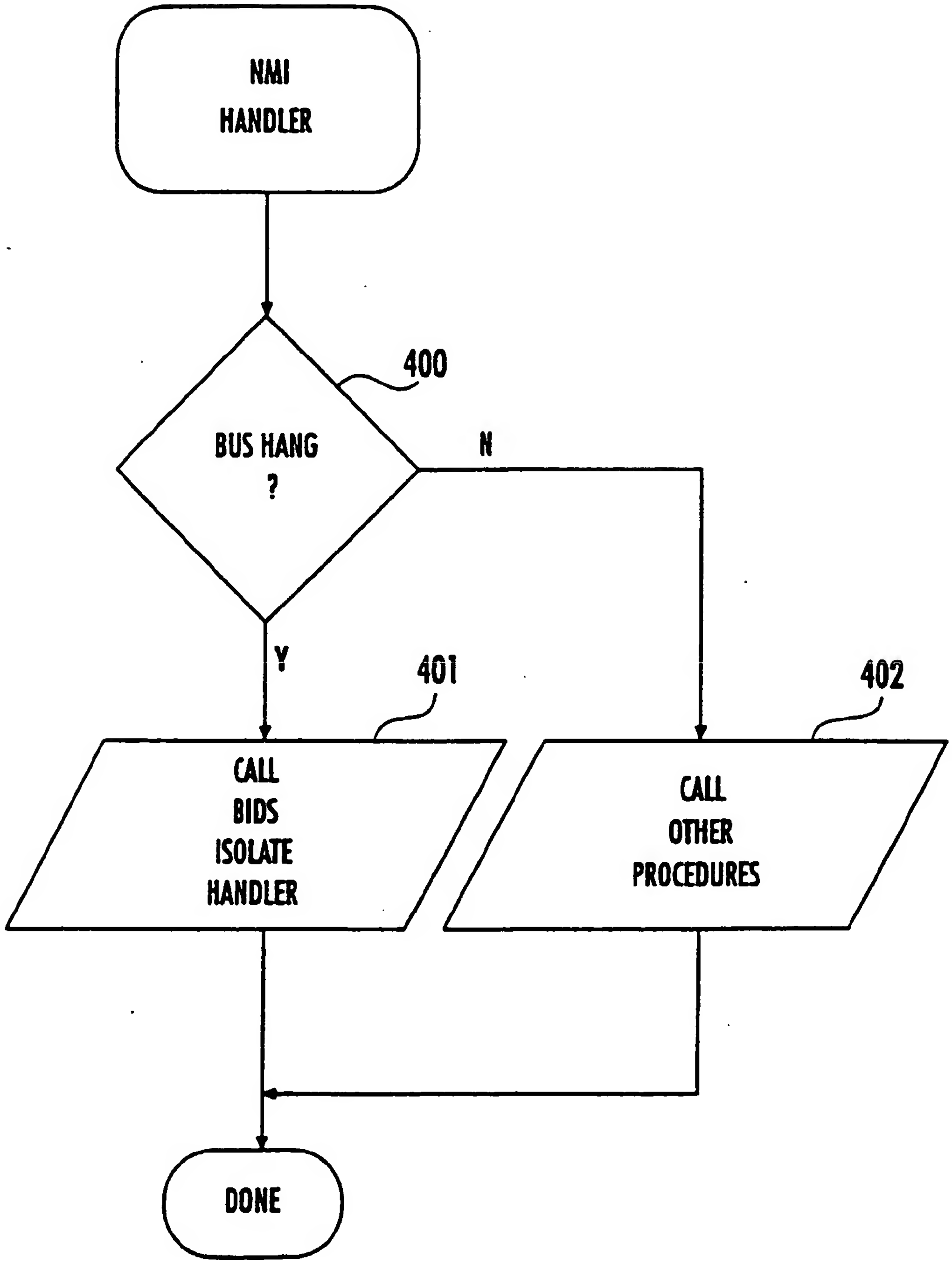

NMI
HANDLER
400
BUS HANG
?
N
Y
401
402
CALL
BIDS
ISOLATE
HANDLER
CALL
OTHER
PROCEDURES
DONE

FIG. 37

FIG. 39

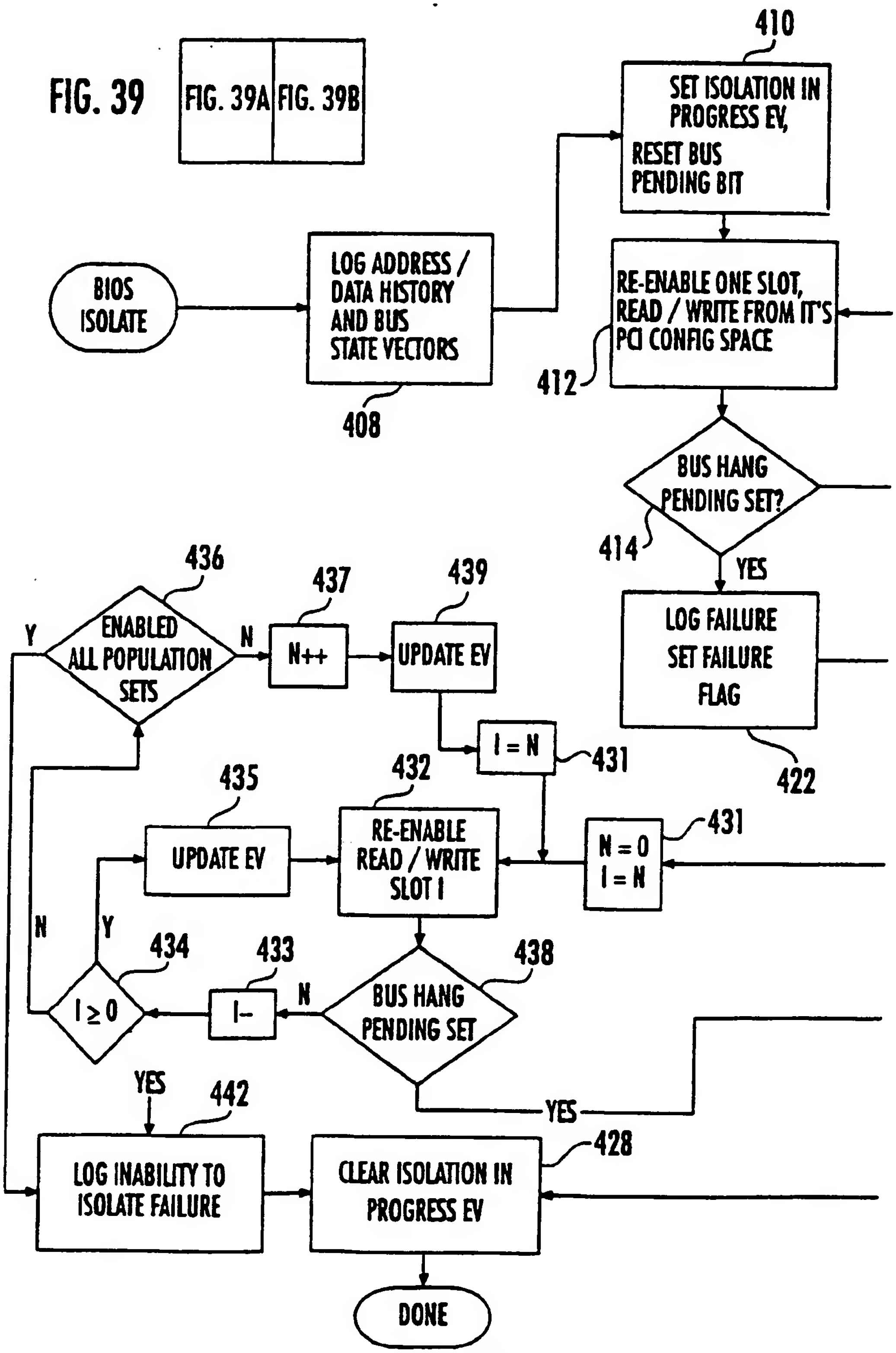


FIG. 39A

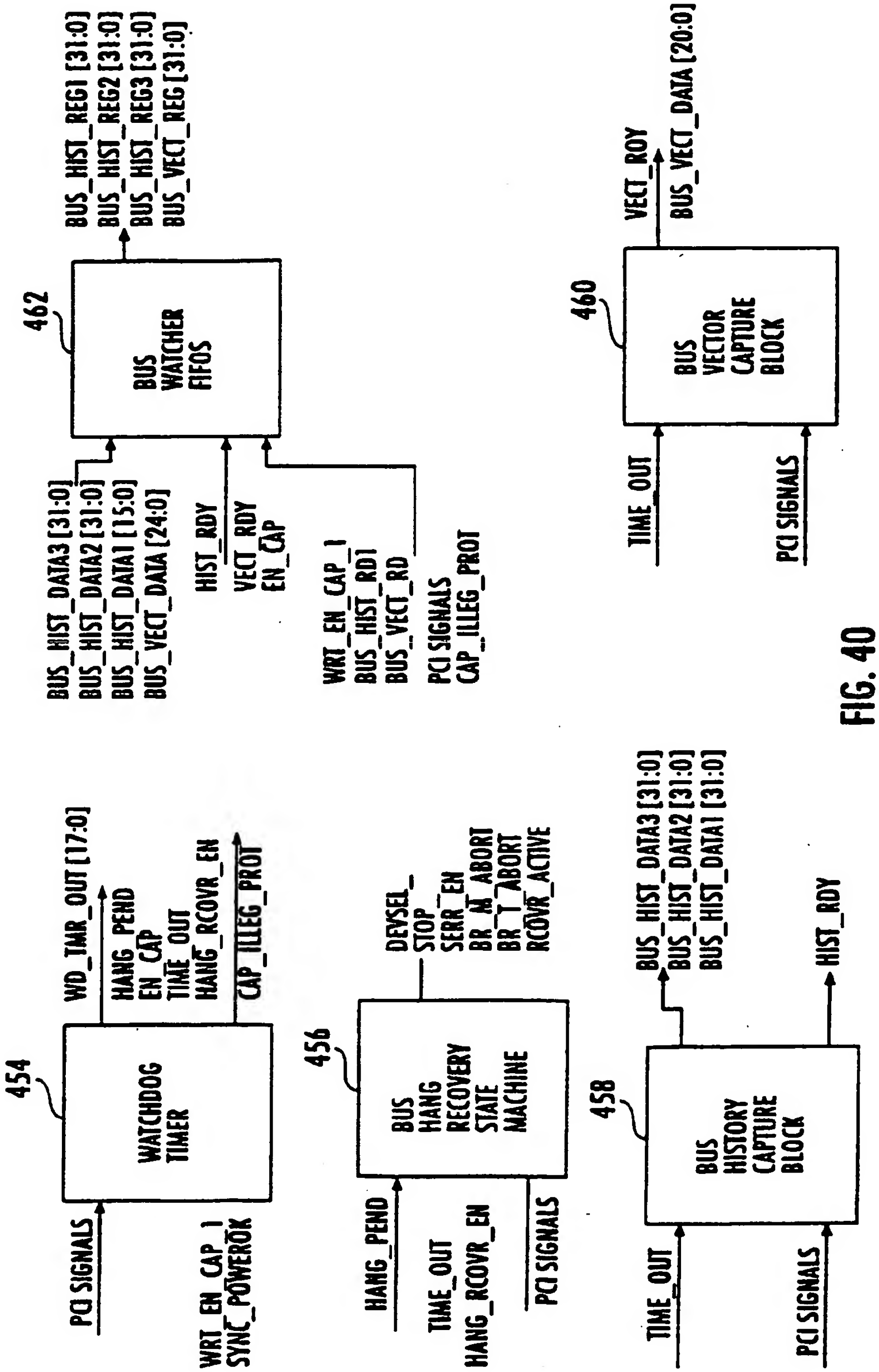

454
WATCHDOG TIMER
PCI SIGNALS
WRT_EN_CAP_1
SYNC_POWEROK
WD_TMR_OUT [17:0]
HANG_PEND
EN_CAP
TIME_OUT
HANG_RCOVR_EN
CAP_ILLEG_PROT
456
BUS HANG RECOVERY STATE MACHINE
HANG_PEND
TIME_OUT
HANG_RCOVR_EN
PCI SIGNALS
DEVSEL_
STOP_
SERR_EN
BR_M_ABORT
BR_T_ABORT
RCOVR_ACTIVE
458
BUS HISTORY CAPTURE BLOCK
TIME_OUT
PCI SIGNALS
BUS_HIST_DATA3 [31:0]
BUS_HIST_DATA2 [31:0]
BUS_HIST_DATA1 [31:0]
HIST_RDY
462
BUS WATCHER FIFOS
BUS_HIST_DATA3 [31:0]
BUS_HIST_DATA2 [31:0]
BUS_HIST_DATA1 [15:0]
BUS_VECT_DATA [24:0]
HIST_RDY
VECT_RDY
EN_CAP
WRT_EN_CAP_1
BUS_HIST_RD1
BUS_VECT_RD
PCI SIGNALS
CAP_ILLEG_PROT
BUS_HIST_REG1 [31:0]
BUS_HIST_REG2 [31:0]
BUS_HIST_REG3 [31:0]
BUS_VECT_REG [31:0]
460
BUS VECTOR CAPTURE BLOCK
TIME_OUT
PCI SIGNALS
VECT_RDY
BUS_VECT_DATA [20:0]

FIG. 40

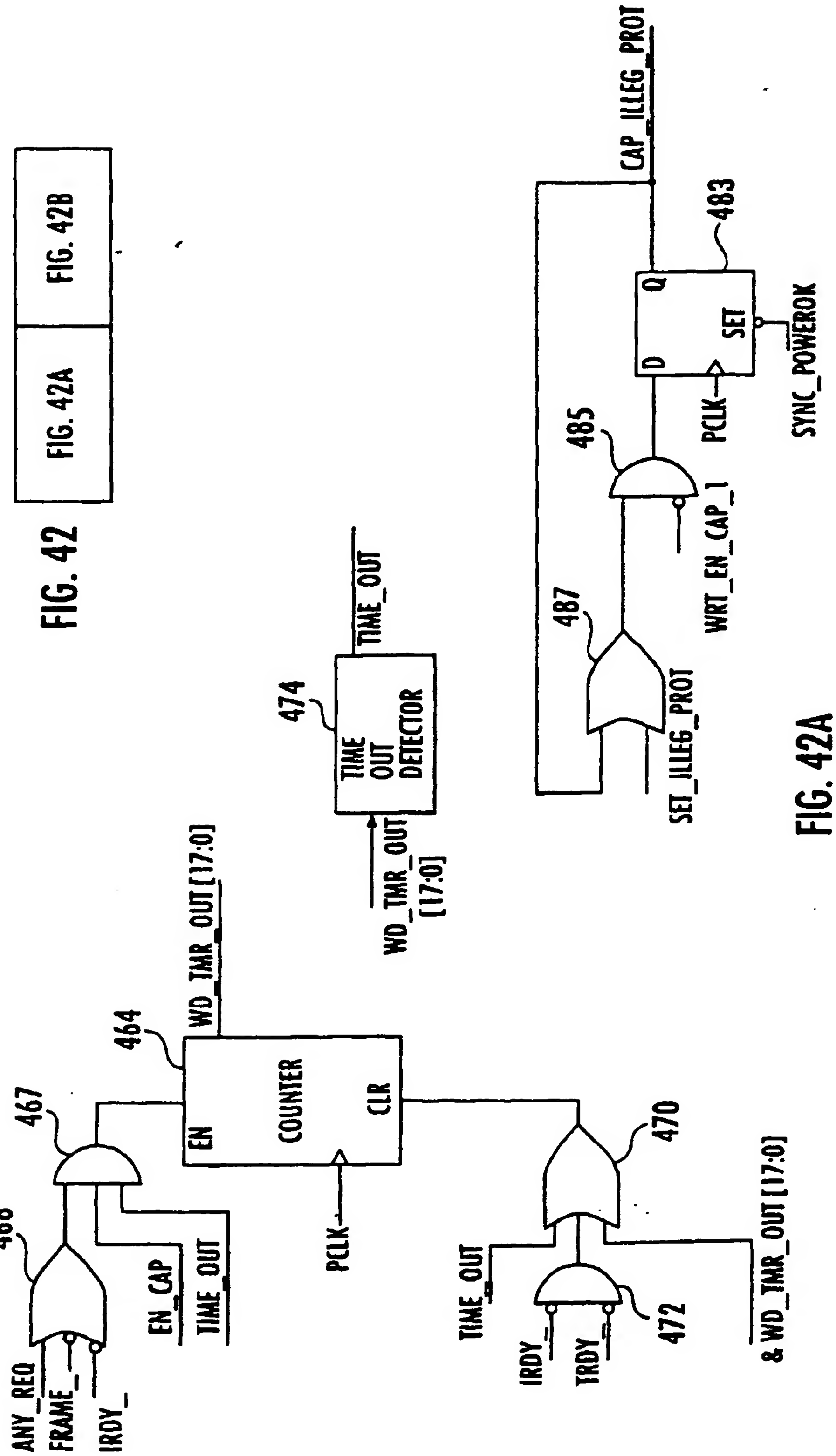


FIG. 42

FIG. 42A

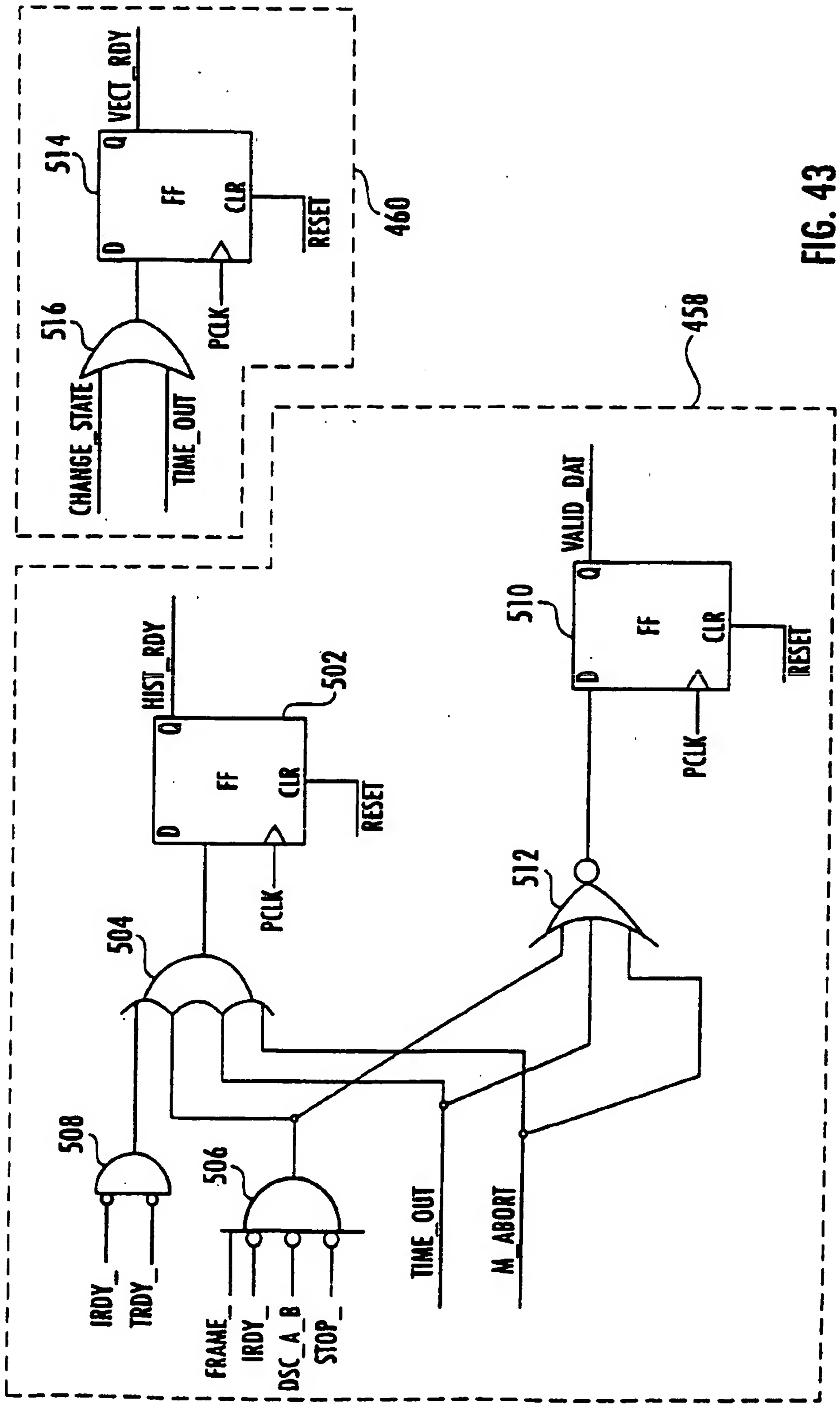

VECT_RDY
514
Q
FF
CLR
D
RESET
460
516
PCLK
CHANGE_STATE
TIME_OUT
458
VALID_DAT
510
Q
FF
CLR
D
RESET
PCLK
HIST_RDY
502
Q
FF
CLR
D
RESET
PCLK
512
504
508
506
IRDY_
TRDY_
FRAME_
IRDY_
DSC_A_B
STOP_
TIME_OUT
M_ABORT

FIG. 43

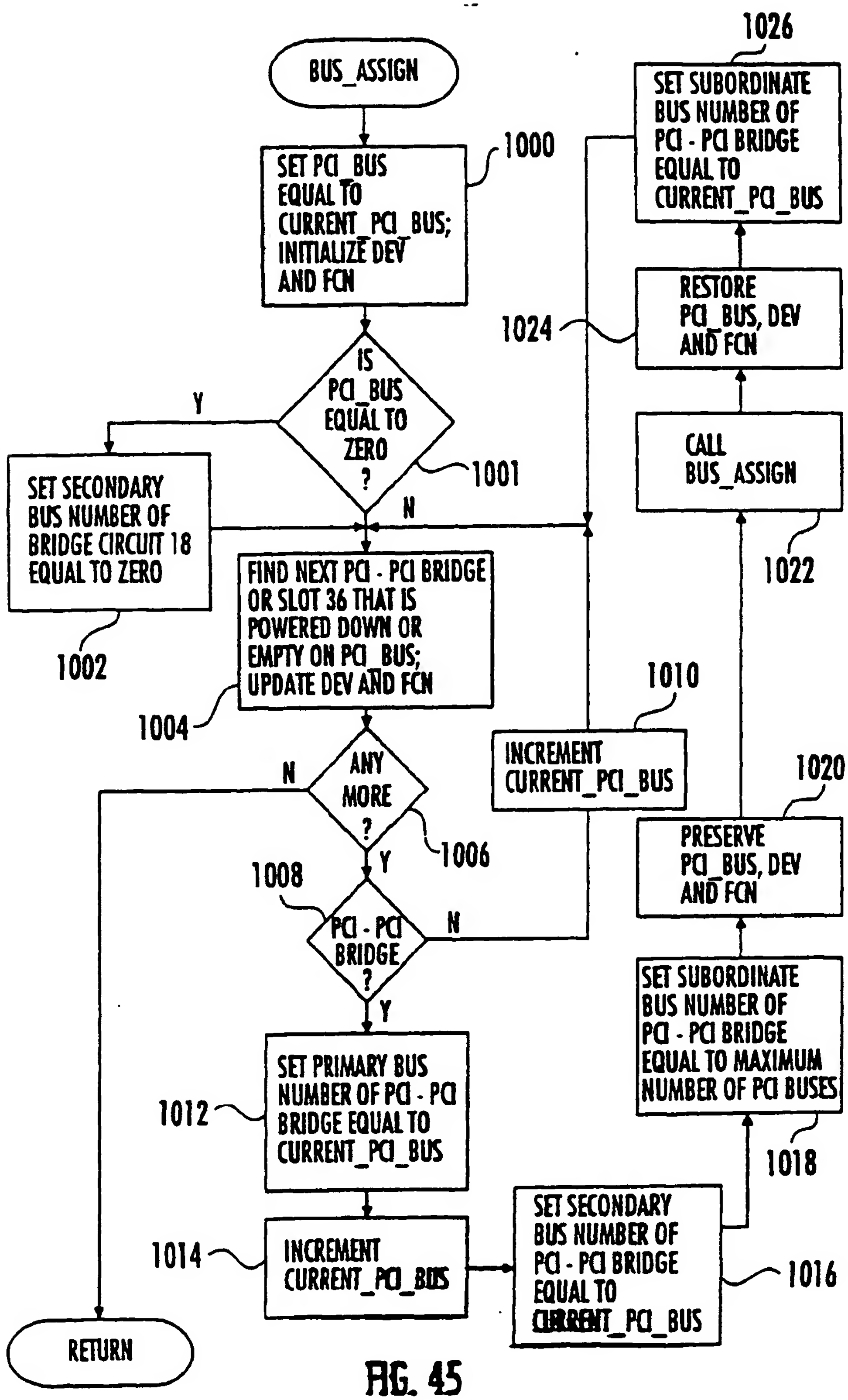

BUS_ASSIGN
SET PCI_BUS EQUAL TO CURRENT_PCI_BUS; INITIALIZE DEV AND FCN
1000
IS PCI_BUS EQUAL TO ZERO ?
1001
Y
N
SET SECONDARY BUS NUMBER OF BRIDGE CIRCUIT 18 EQUAL TO ZERO
1002
FIND NEXT PCI - PCI BRIDGE OR SLOT 36 THAT IS POWERED DOWN OR EMPTY ON PCI_BUS; UPDATE DEV AND FCN
1004
ANY MORE ?
1006
N
Y
PCI - PCI BRIDGE ?
1008
N
Y
INCREMENT CURRENT_PCI_BUS
1010
SET PRIMARY BUS NUMBER OF PCI - PCI BRIDGE EQUAL TO CURRENT_PCI_BUS
1012
INCREMENT CURRENT_PCI_BUS
1014
SET SECONDARY BUS NUMBER OF PCI - PCI BRIDGE EQUAL TO CURRENT_PCI_BUS
1016
SET SUBORDINATE BUS NUMBER OF PCI - PCI BRIDGE EQUAL TO MAXIMUM NUMBER OF PCI BUSES
1018
PRESERVE PCI_BUS, DEV AND FCN
1020
CALL BUS_ASSIGN
1022
RESTORE PCI_BUS, DEV AND FCN
1024
SET SUBORDINATE BUS NUMBER OF PCI - PCI BRIDGE EQUAL TO CURRENT_PCI_BUS
1026
RETURN

FIG. 45

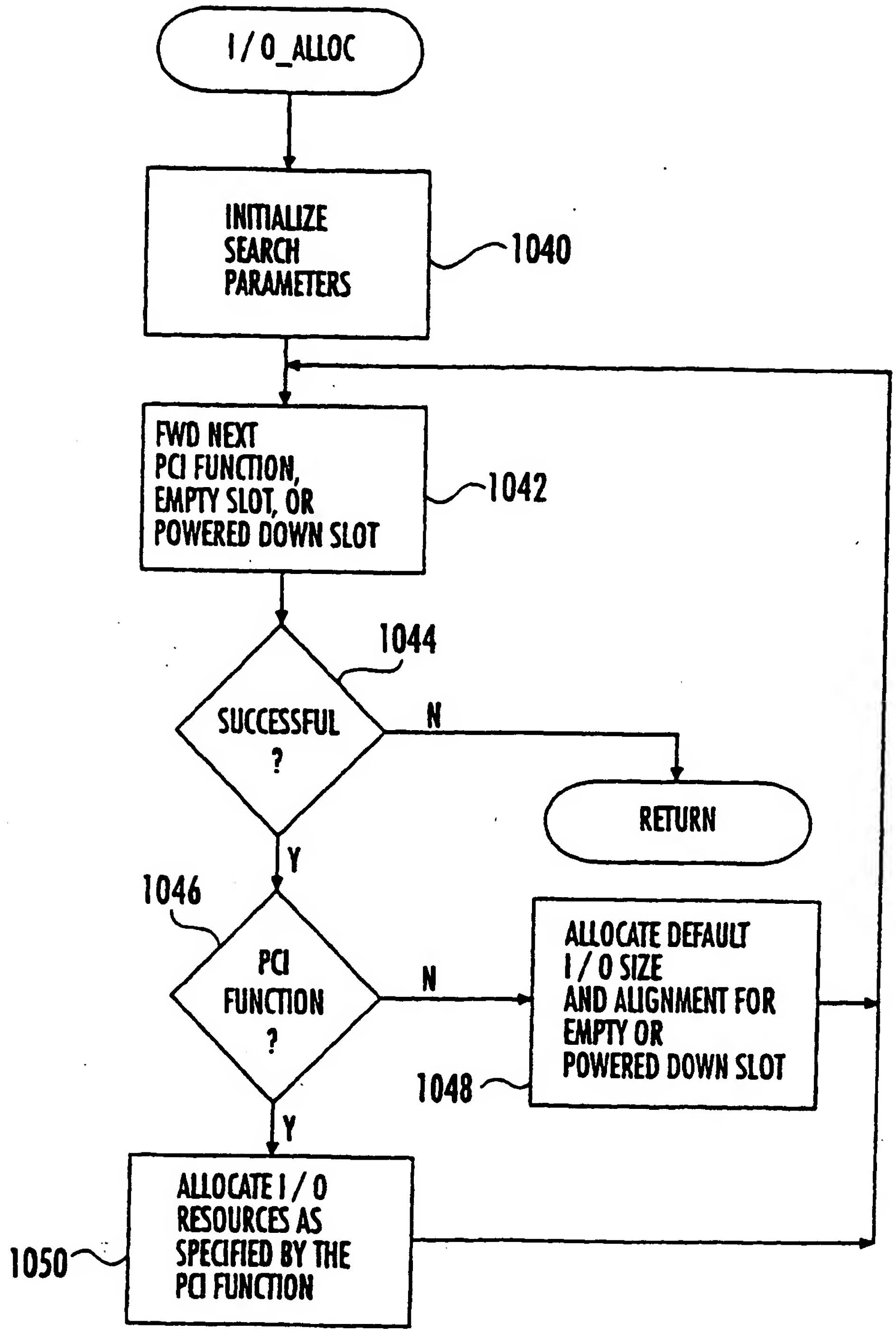

I/O_ALLOC
INITIALIZE SEARCH PARAMETERS
1040
FWD NEXT PCI FUNCTION, EMPTY SLOT, OR POWERED DOWN SLOT
1042
1044
SUCCESSFUL ?
N
RETURN
Y
1046
PCI FUNCTION ?
N
ALLOCATE DEFAULT I/O SIZE AND ALIGNMENT FOR EMPTY OR POWERED DOWN SLOT
1048
Y
ALLOCATE I/O RESOURCES AS SPECIFIED BY THE PCI FUNCTION
1050

FIG. 47

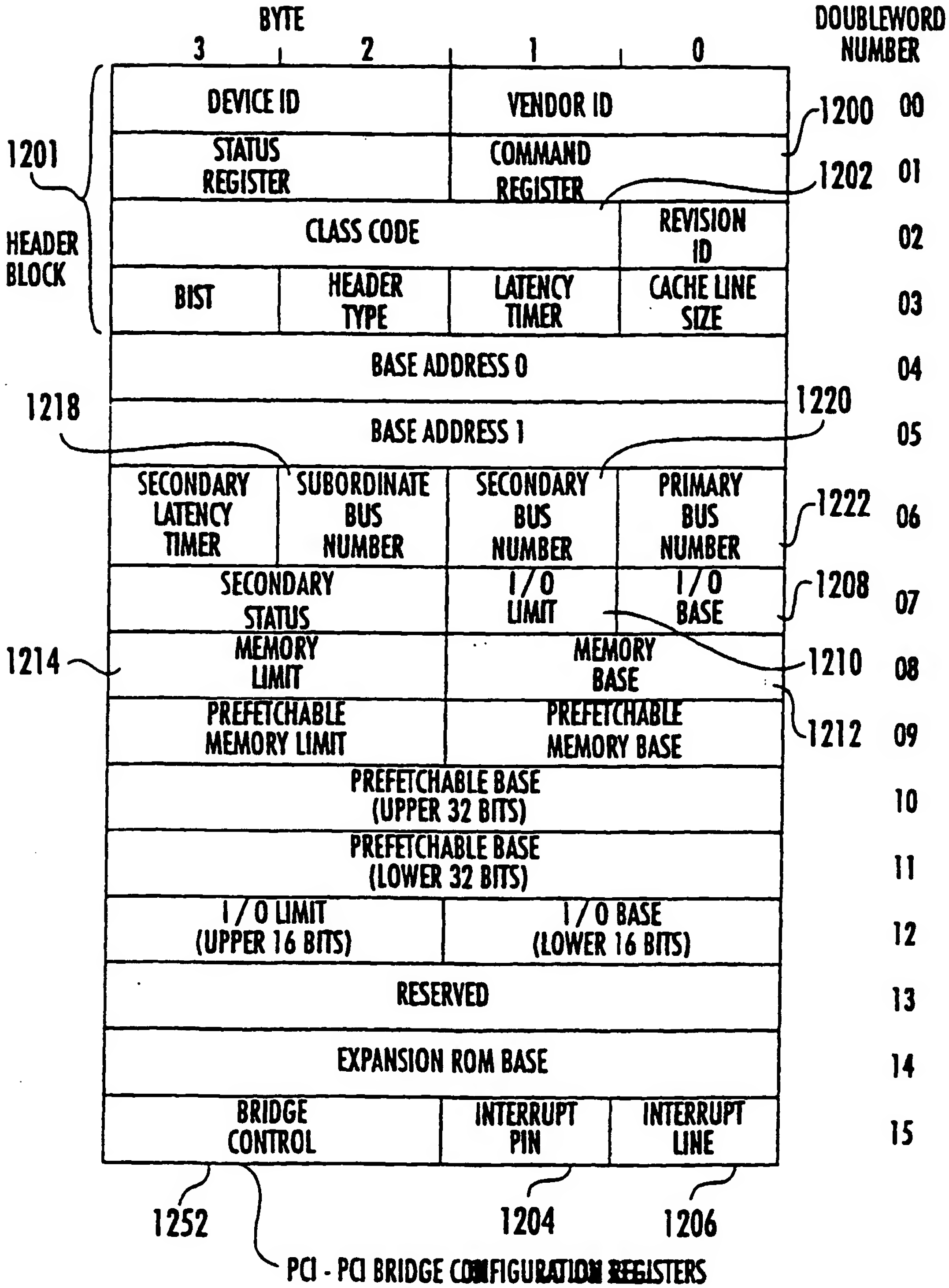

BYTE
3
2
1
0
DOUBLEWORD NUMBER
DEVICE ID
VENDOR ID
1200
00
1201
STATUS REGISTER
COMMAND REGISTER
1202
01
HEADER BLOCK
CLASS CODE
REVISION ID
02
BIST
HEADER TYPE
LATENCY TIMER
CACHE LINE SIZE
03
BASE ADDRESS 0
04
1218
BASE ADDRESS 1
1220
05
SECONDARY LATENCY TIMER
SUBORDINATE BUS NUMBER
SECONDARY BUS NUMBER
PRIMARY BUS NUMBER
1222
06
SECONDARY STATUS
I / O LIMIT
I / O BASE
1208
07
1214
MEMORY LIMIT
MEMORY BASE
1210
08
PREFETCHABLE MEMORY LIMIT
PREFETCHABLE MEMORY BASE
1212
09
PREFETCHABLE BASE (UPPER 32 BITS)
10
PREFETCHABLE BASE (LOWER 32 BITS)
11
I / O LIMIT (UPPER 16 BITS)
I / O BASE (LOWER 16 BITS)
12
RESERVED
13
EXPANSION ROM BASE
14
BRIDGE CONTROL
INTERRUPT PIN
INTERRUPT LINE
15
1252
1204
1206
PCI - PCI BRIDGE CONFIGURATION REGISTERS

FIG. 49

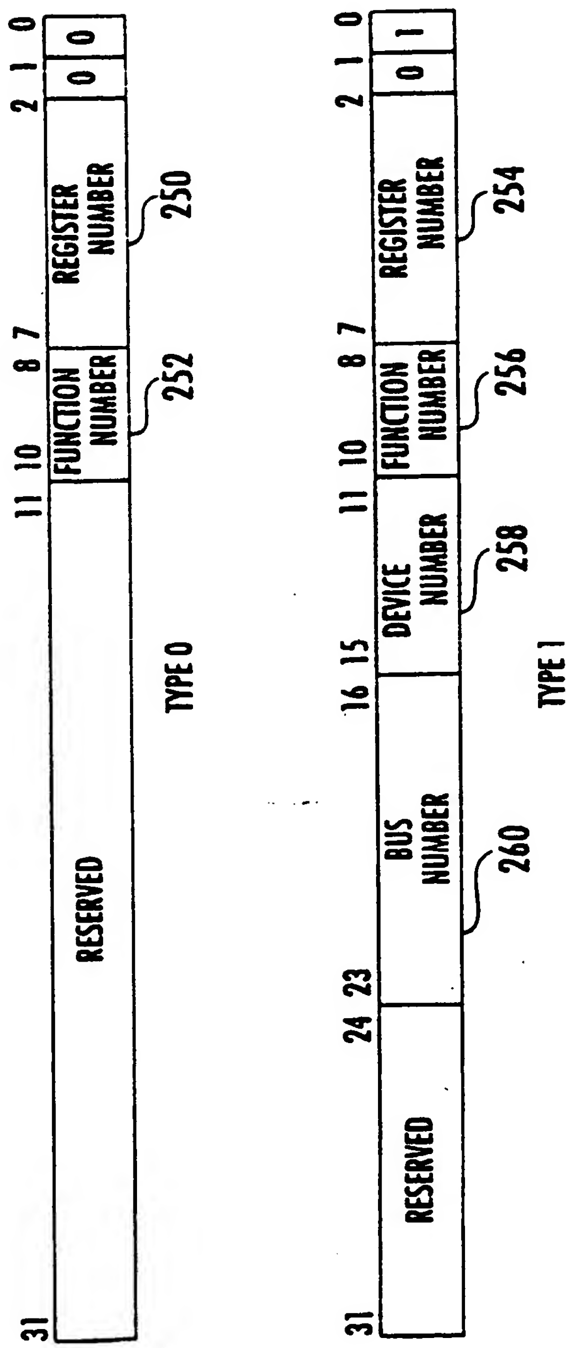

31
RESERVED
11 10
FUNCTION NUMBER
8 7
REGISTER NUMBER
2 1 0
0
0
250
252
TYPE 0
31
RESERVED
24 23
BUS NUMBER
16 15
DEVICE NUMBER
11 10
FUNCTION NUMBER
8 7
REGISTER NUMBER
2 1 0
0
1
254
256
258
260
TYPE 1

FIG. 51

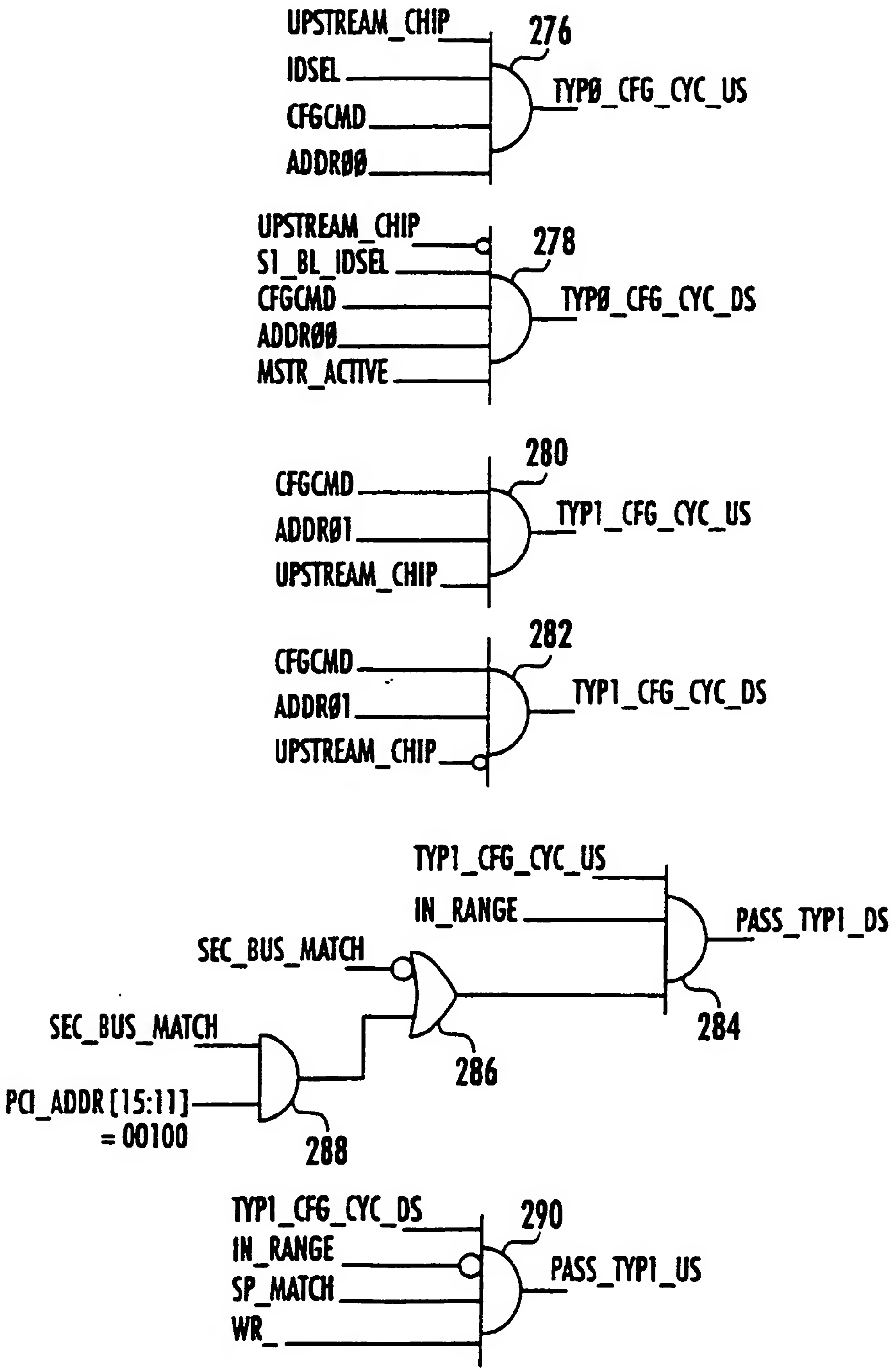

UPSTREAM_CHIP
IDSEL
CFGCMD
ADDR00
276
TYP0_CFG_CYC_US
UPSTREAM_CHIP
S1_BL_IDSEL
CFGCMD
ADDR00
MSTR_ACTIVE
278
TYP0_CFG_CYC_DS
CFGCMD
ADDR01
UPSTREAM_CHIP
280
TYP1_CFG_CYC_US
CFGCMD
ADDR01
UPSTREAM_CHIP
282
TYP1_CFG_CYC_DS
TYP1_CFG_CYC_US
IN_RANGE
PASS_TYP1_DS
SEC_BUS_MATCH
284
SEC_BUS_MATCH
286
PCI_ADDR [15:11]
= 00100
288
TYP1_CFG_CYC_DS
IN_RANGE
SP_MATCH
WR_
290
PASS_TYP1_US

FIG. 53A

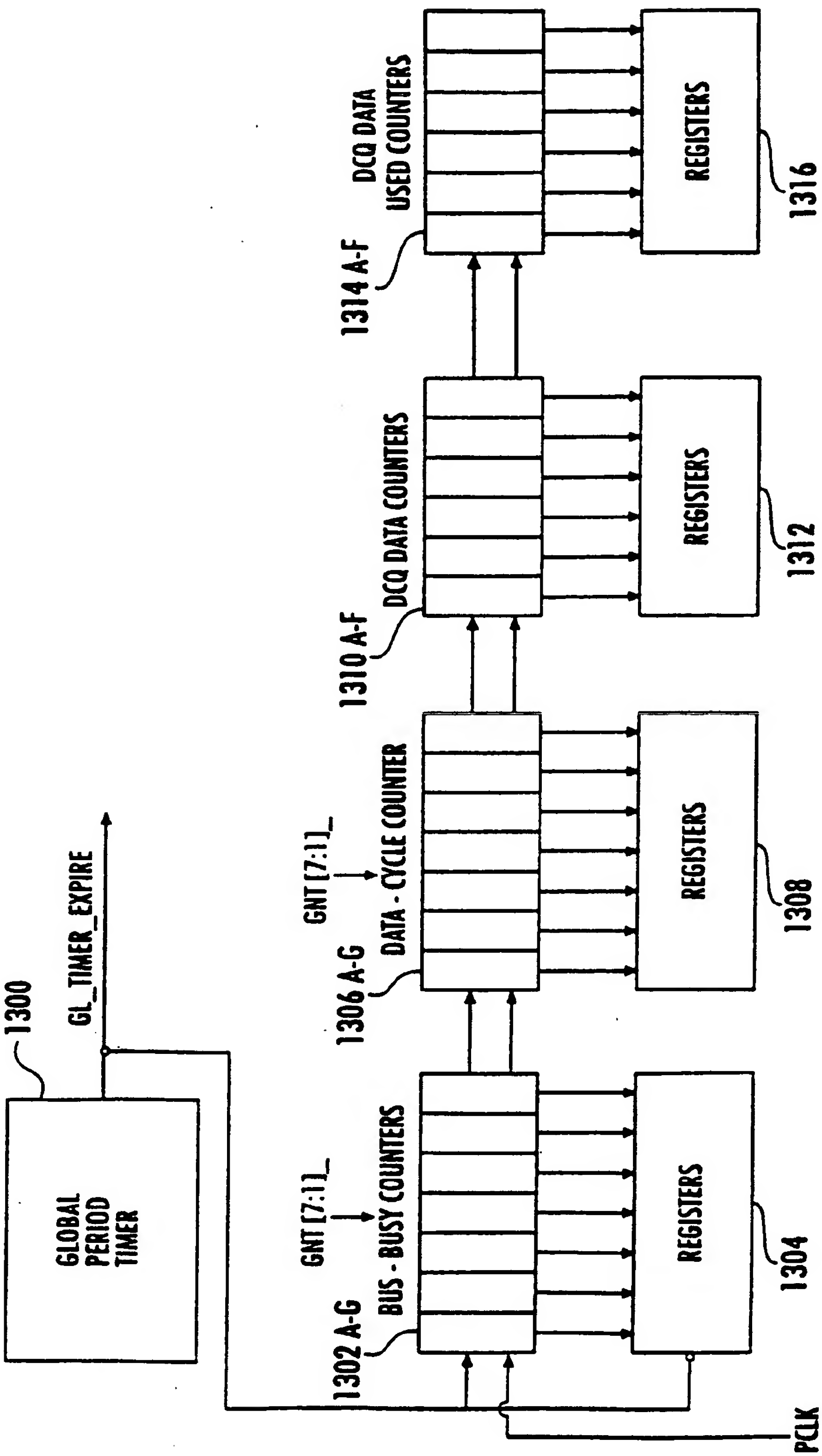

GLOBAL PERIOD TIMER
1300
GL_TIMER_EXPIRE
1302 A-G
BUS - BUSY COUNTERS
GNT [7:1]_
REGISTERS
1304
1306 A-G
DATA - CYCLE COUNTER
GNT [7:1]_
REGISTERS
1308
1310 A-F
DCQ DATA COUNTERS
REGISTERS
1312
1314 A-F
DCQ DATA USED COUNTERS
REGISTERS
1316
PCLK

FIG. 54A

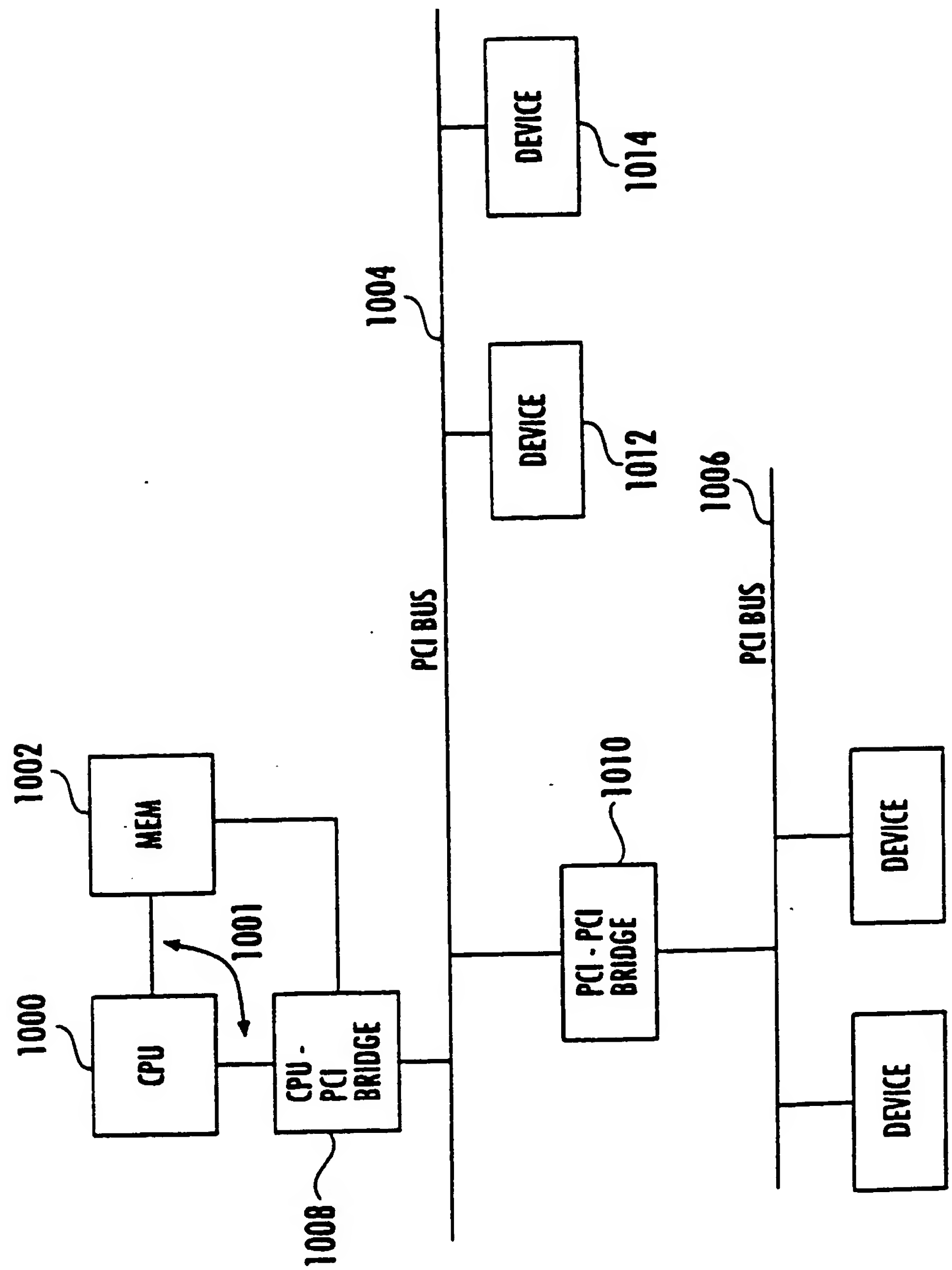


FIG. 55 (PRIOR ART)

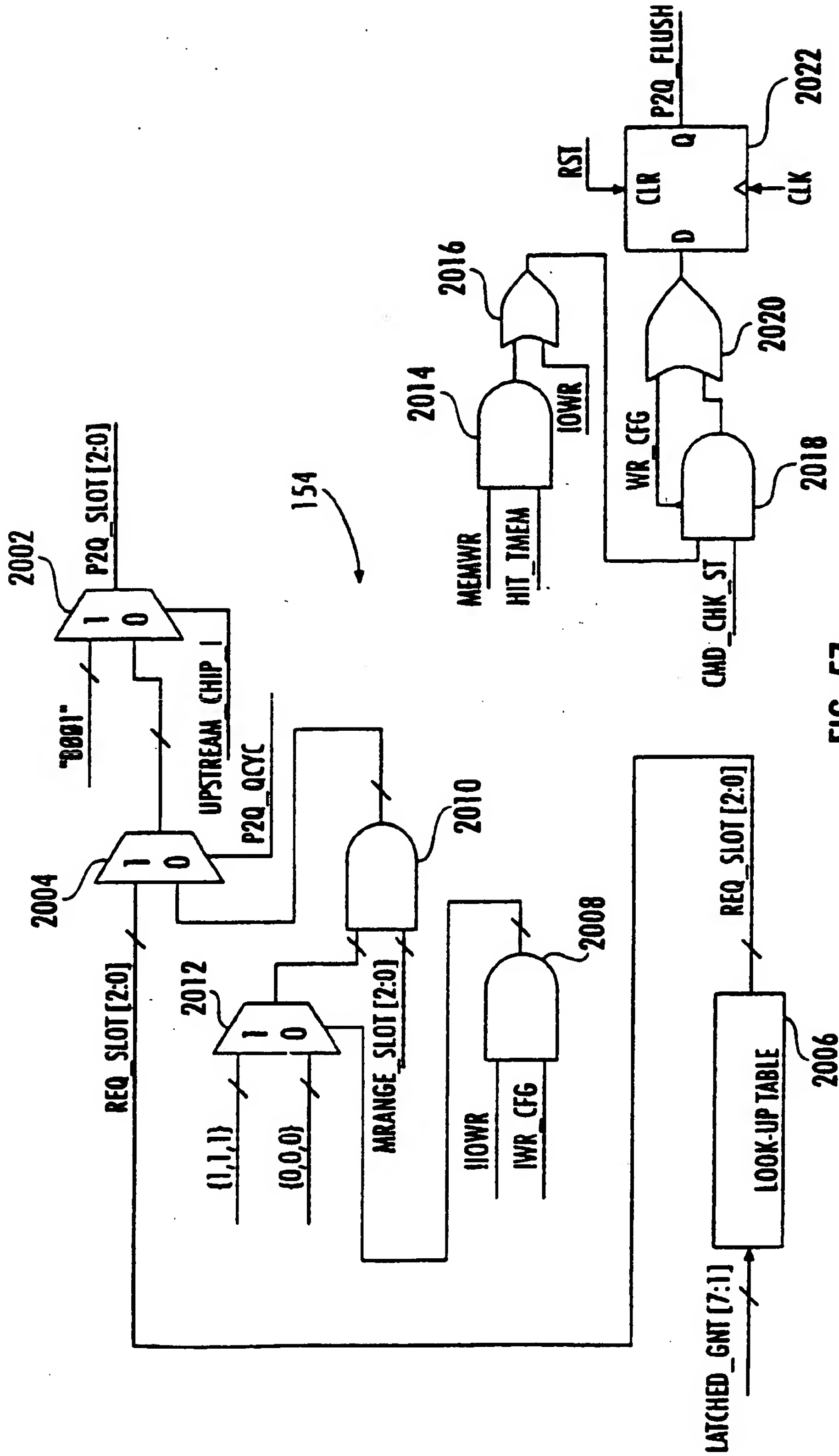

2002
P2Q_SLOT [2:0]
1
0
"B001"
UPSTREAM_CHIP_I
P2Q_QCYC
2004
REQ_SLOT [2:0]
154
2010
2012
{1,1,1}
{0,0,0}
MRANGE_SLOT [2:0]
2008
IIOWR
IWR_CFG
LOOK-UP TABLE
2006
LATCHED_GNT [7:1]
REQ_SLOT [2:0]
MEMWR
HIT_TMEM
2014
IOWR
2016
WR_CFG
2018
2020
CMD_CHK_ST
RST
CLR
Q
D
CLK
2022
P2Q_FLUSH

FIG. 57

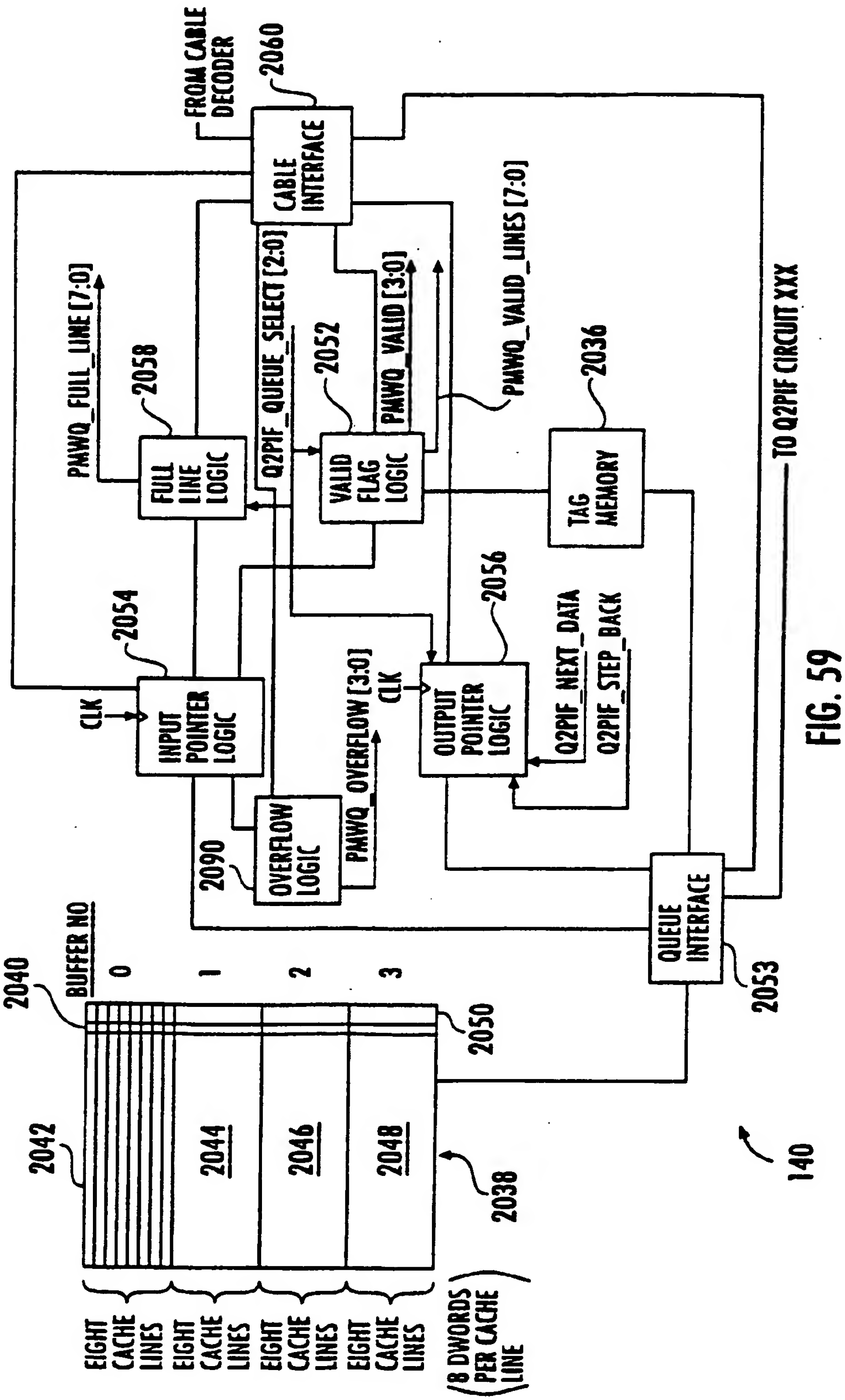

FROM CABLE DECODER
2060
CABLE INTERFACE
PMWQ_FULL_LINE [7:0]
2058
FULL LINE LOGIC
Q2PIF_QUEUE_SELECT [2:0]
2052
PMWQ_VALID [3:0]
VALID FLAG LOGIC
PMWQ_VALID_LINES [7:0]
2036
TAG MEMORY
TO Q2PIF CIRCUIT XXX
2054
CLK
INPUT POINTER LOGIC
PMWQ_OVERFLOW [3:0]
CLK
2056
OUTPUT POINTER LOGIC
Q2PIF_NEXT_DATA
Q2PIF_STEP_BACK
2090
OVERFLOW LOGIC
QUEUE INTERFACE
2053
2040
BUFFER NO
0
1
2
3
2050
2042
2044
2046
2048
2038
EIGHT CACHE LINES
EIGHT CACHE LINES
EIGHT CACHE LINES
EIGHT CACHE LINES
(8 DWORDS PER CACHE LINE)
140

FIG. 59

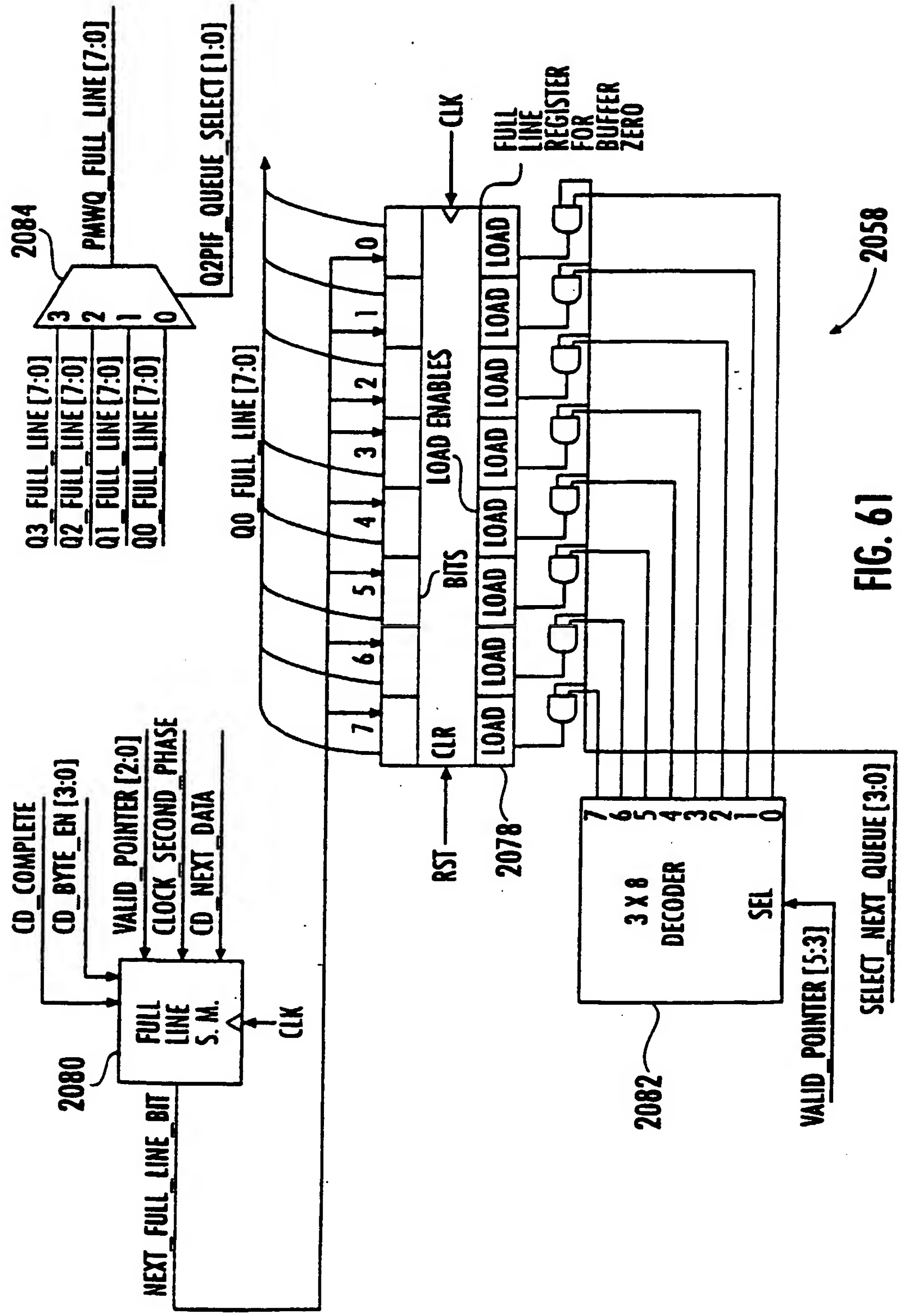

2084
PMWQ_FULL_LINE [7:0]
Q2PIF_QUEUE_SELECT [1:0]
Q3_FULL_LINE [7:0]
Q2_FULL_LINE [7:0]
Q1_FULL_LINE [7:0]
Q0_FULL_LINE [7:0]
3
2
1
0
Q0_FULL_LINE [7:0]
CLK
FULL LINE REGISTER FOR BUFFER ZERO
LOAD ENABLES
BITS
CLR
LOAD
RST
2078
2058
CD_COMPLETE
CD_BYTE_EN [3:0]
VALID_POINTER [2:0]
CLOCK_SECOND_PHASE
CD_NEXT_DATA
FULL LINE S.M.
CLK
2080
NEXT_FULL_LINE_BIT
3 X 8 DECODER
SEL
2082
VALID_POINTER [5:3]
SELECT_NEXT_QUEUE [3:0]

FIG. 61

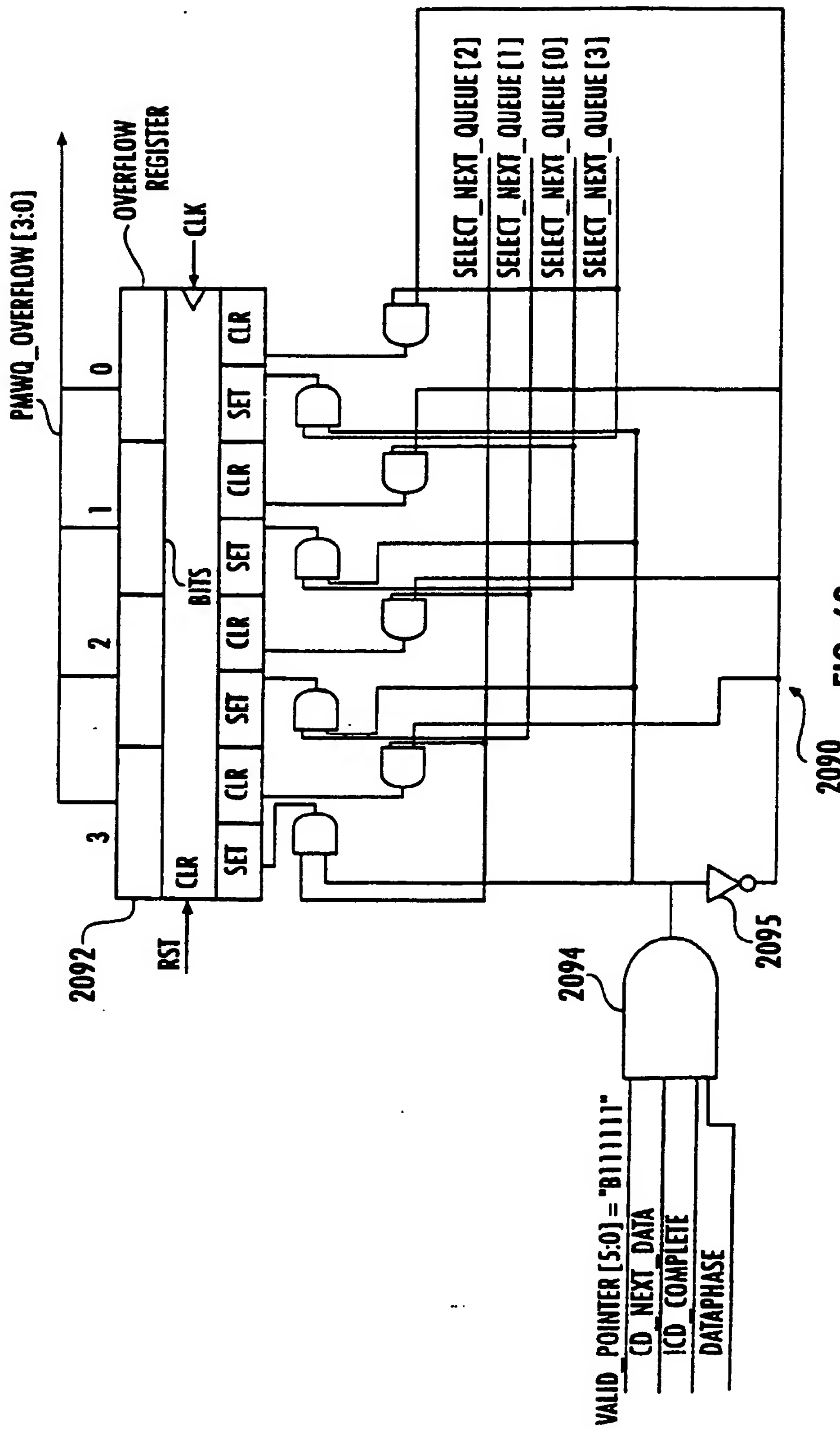

PMWQ_OVERFLOW [3:0]
OVERFLOW REGISTER
CLK
BITS
RST
CLR
SET
0
1
2
3
SELECT_NEXT_QUEUE [2]
SELECT_NEXT_QUEUE [1]
SELECT_NEXT_QUEUE [0]
SELECT_NEXT_QUEUE [3]
2092
2094
2095
2090
VALID_POINTER [5:0] = "B111111"
CD_NEXT_DATA
ICD_COMPLETE
DATAPHASE

FIG. 63

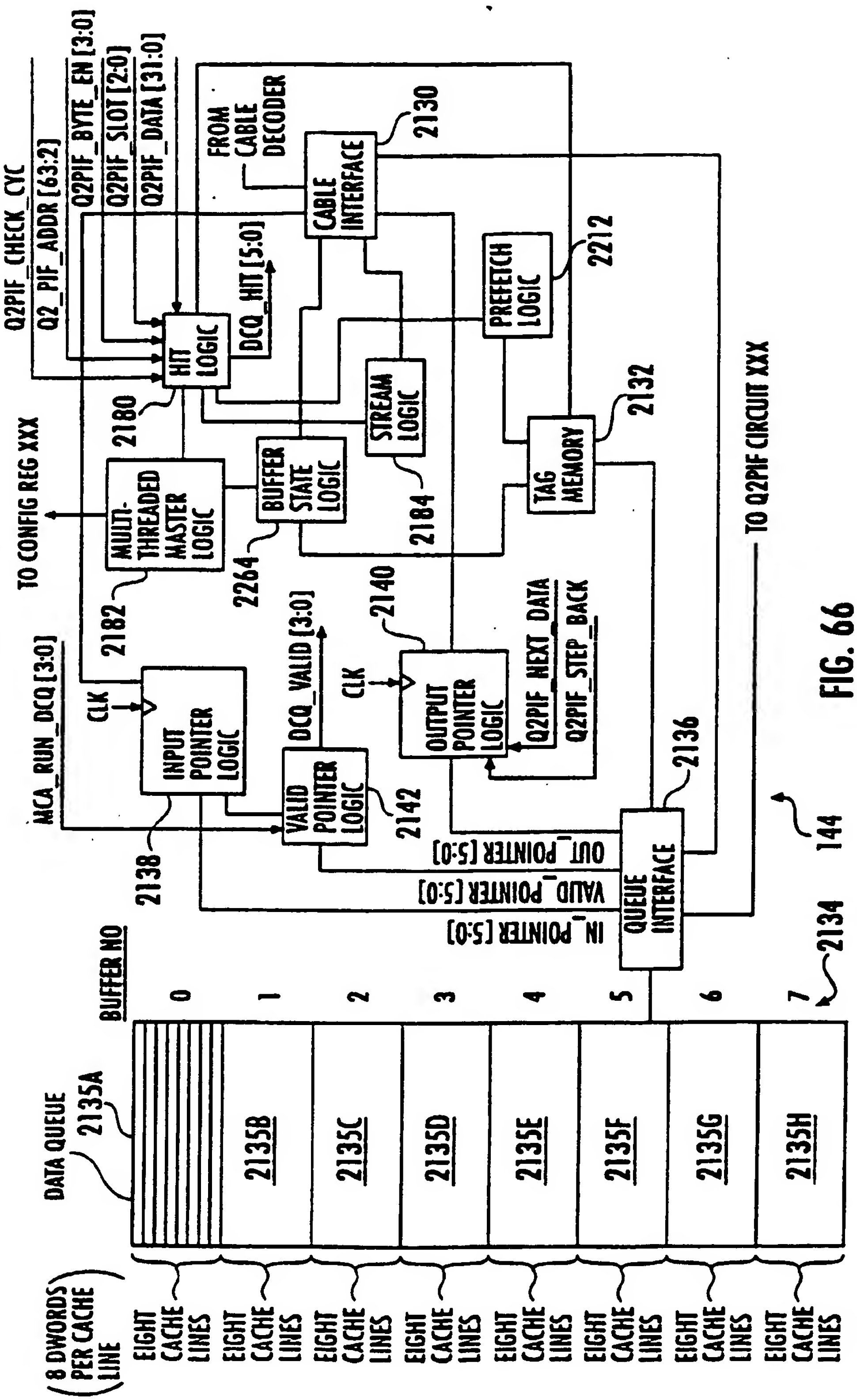

Q2PIF_CHECK_CYC
Q2_PIF_ADDR [63:2]
Q2PIF_BYTE_EN [3:0]
Q2PIF_SLOT [2:0]
Q2PIF_DATA [31:0]
FROM CABLE DECODER
CABLE INTERFACE
2130
HIT LOGIC
DCQ_HIT [5:0]
PREFETCH LOGIC
2212
2180
STREAM LOGIC
TO CONFIG REG XXX
MULTI-THREADED MASTER LOGIC
BUFFER STATE LOGIC
TAG MEMORY
2132
2184
TO Q2PIF CIRCUIT XXX
2182
2264
2140
DCQ_VALID [3:0]
Q2PIF_NEXT_DATA
Q2PIF_STEP_BACK
MCA_RUN_DCQ [3:0]
CLK
INPUT POINTER LOGIC
CLK
OUTPUT POINTER LOGIC
VALID POINTER LOGIC
2142
2136
2138
OUT_POINTER [5:0]
VALID_POINTER [5:0]
IN_POINTER [5:0]
QUEUE INTERFACE
144
2134
BUFFER NO
0
1
2
3
4
5
6
7
DATA QUEUE
2135A
2135B
2135C
2135D
2135E
2135F
2135G
2135H
(8 DWORDS PER CACHE LINE)
EIGHT CACHE LINES
EIGHT CACHE LINES
EIGHT CACHE LINES
EIGHT CACHE LINES
EIGHT CACHE LINES
EIGHT CACHE LINES
EIGHT CACHE LINES
EIGHT CACHE LINES

FIG. 66

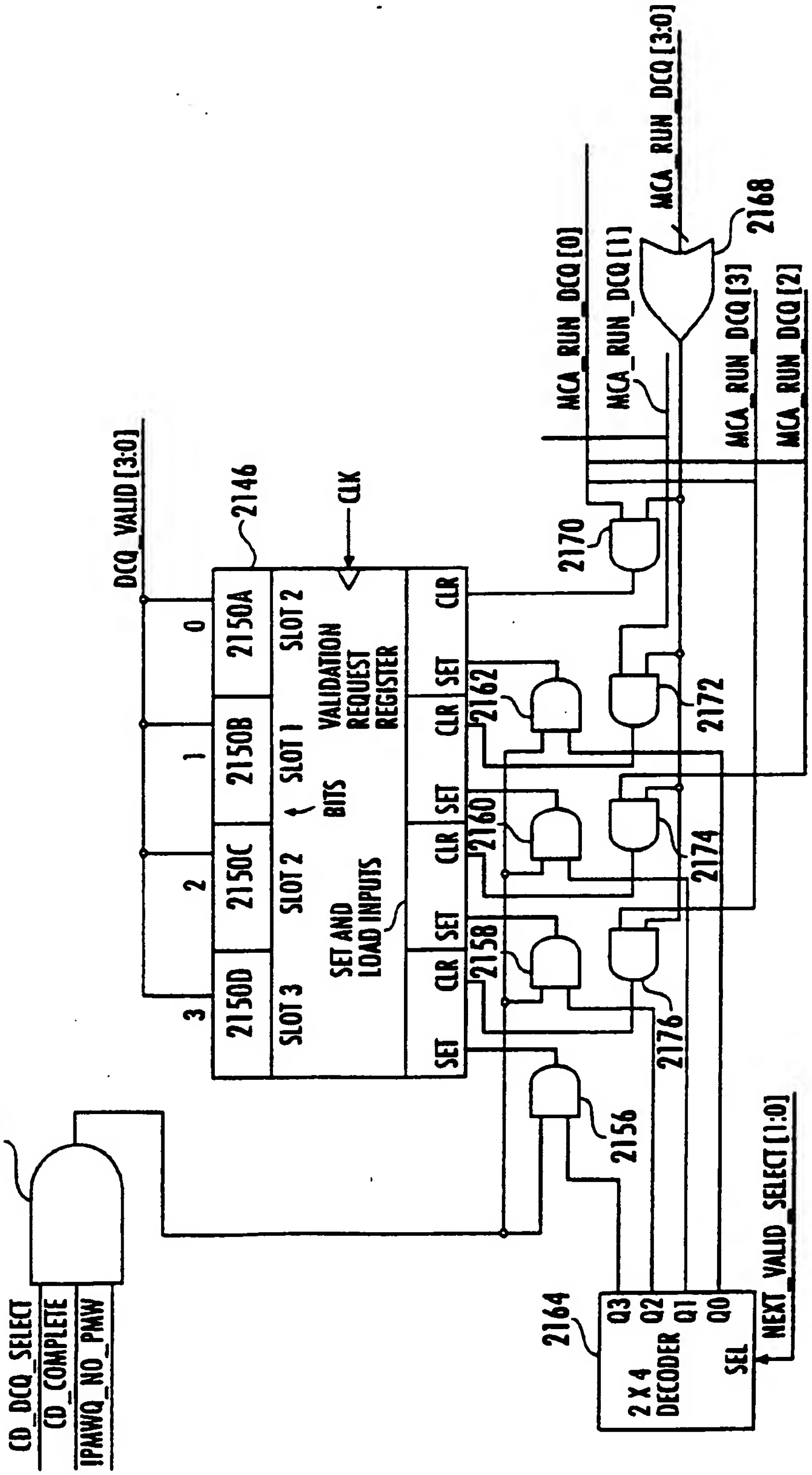
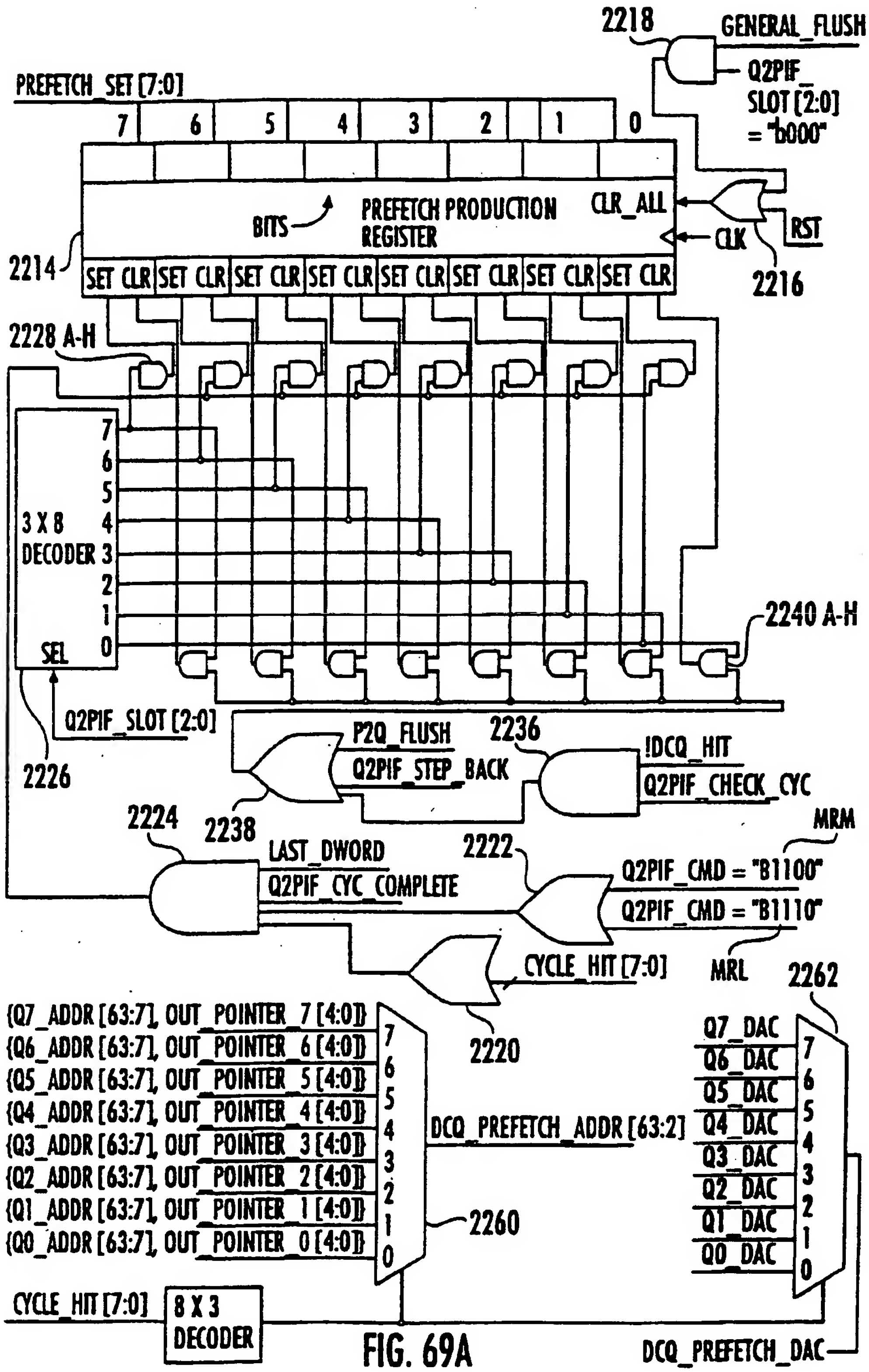

DCQ_VALID [3:0]
2146
CLK
0
1
2
3
2150A
2150B
2150C
2150D
SLOT 2
SLOT 1
SLOT 2
SLOT 3
VALIDATION
REQUEST
REGISTER
BITS
SET AND
LOAD INPUTS
CLR
SET
2154
2156
2158
2160
2162
2164
2170
2172
2174
2176
2168
CD_DCQ_SELECT
CD_COMPLETE
!PMWQ_NO_PMW
2 X 4
DECODER
Q3
Q2
Q1
Q0
SEL
NEXT_VALID_SELECT [1:0]
MCA_RUN_DCQ [0]
MCA_RUN_DCQ [1]
MCA_RUN_DCQ [3]
MCA_RUN_DCQ [2]
MCA_RUN_DCQ [3:0]

FIG. 67B



2218
GENERAL_FLUSH
Q2PIF_SLOT [2:0] = "b000"
PREFETCH_SET [7:0]
7
6
5
4
3
2
1
0
BITS
PREFETCH PRODUCTION REGISTER
CLR_ALL
CLK
RST
2214
2216
SET CLR
2228 A-H
3 X 8 DECODER
SEL
2240 A-H
Q2PIF_SLOT [2:0]
2226
P2Q_FLUSH
Q2PIF_STEP_BACK
2236
!DCQ_HIT
Q2PIF_CHECK_CYC
2224
2238
LAST_DWORD
Q2PIF_CYC_COMPLETE
2222
MRM
Q2PIF_CMD = "B1100"
Q2PIF_CMD = "B1110"
CYCLE_HIT [7:0]
MRL
2262
2220
{Q7_ADDR [63:7], OUT_POINTER_7 [4:0]}
{Q6_ADDR [63:7], OUT_POINTER_6 [4:0]}
{Q5_ADDR [63:7], OUT_POINTER_5 [4:0]}
{Q4_ADDR [63:7], OUT_POINTER_4 [4:0]}
{Q3_ADDR [63:7], OUT_POINTER_3 [4:0]}
{Q2_ADDR [63:7], OUT_POINTER_2 [4:0]}
{Q1_ADDR [63:7], OUT_POINTER_1 [4:0]}
{Q0_ADDR [63:7], OUT_POINTER_0 [4:0]}
DCQ_PREFETCH_ADDR [63:2]
2260
Q7_DAC
Q6_DAC
Q5_DAC
Q4_DAC
Q3_DAC
Q2_DAC
Q1_DAC
Q0_DAC
CYCLE_HIT [7:0]
8 X 3 DECODER
DCQ_PREFETCH_DAC

FIG. 69A

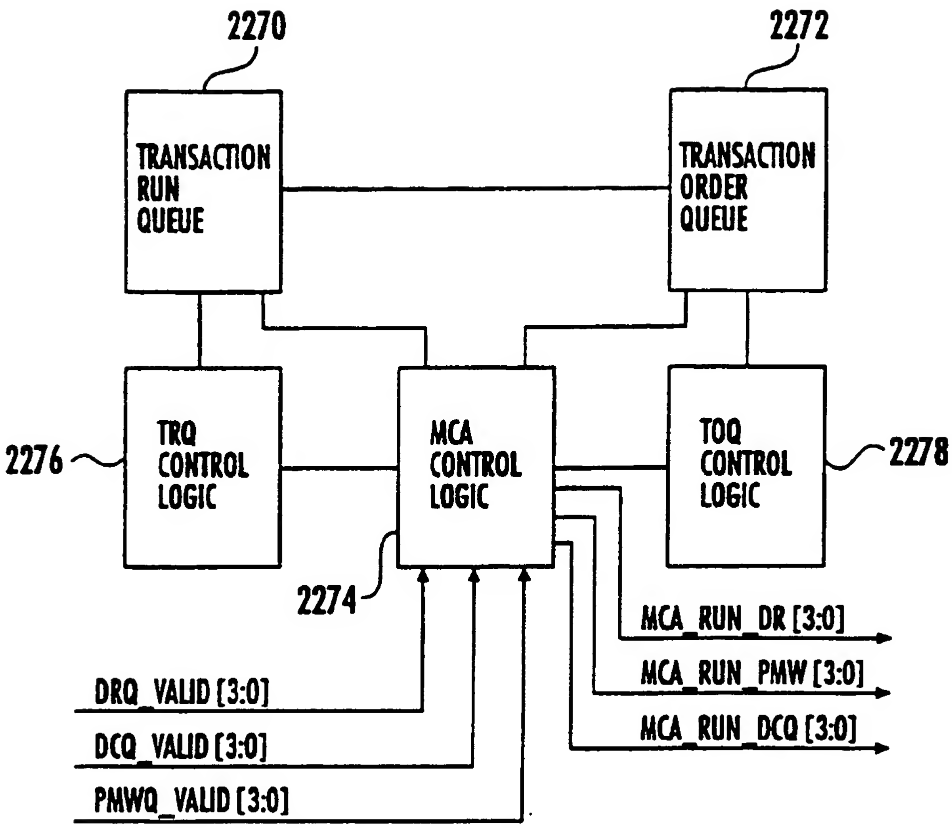

2270
2272
TRANSACTION
RUN
QUEUE
TRANSACTION
ORDER
QUEUE
2276
TRQ
CONTROL
LOGIC
MCA
CONTROL
LOGIC
TOQ
CONTROL
LOGIC
2278
2274
MCA_RUN_DR [3:0]
MCA_RUN_PMW [3:0]
MCA_RUN_DCQ [3:0]
DRQ_VALID [3:0]
DCQ_VALID [3:0]
PMWQ_VALID [3:0]

FIG. 70

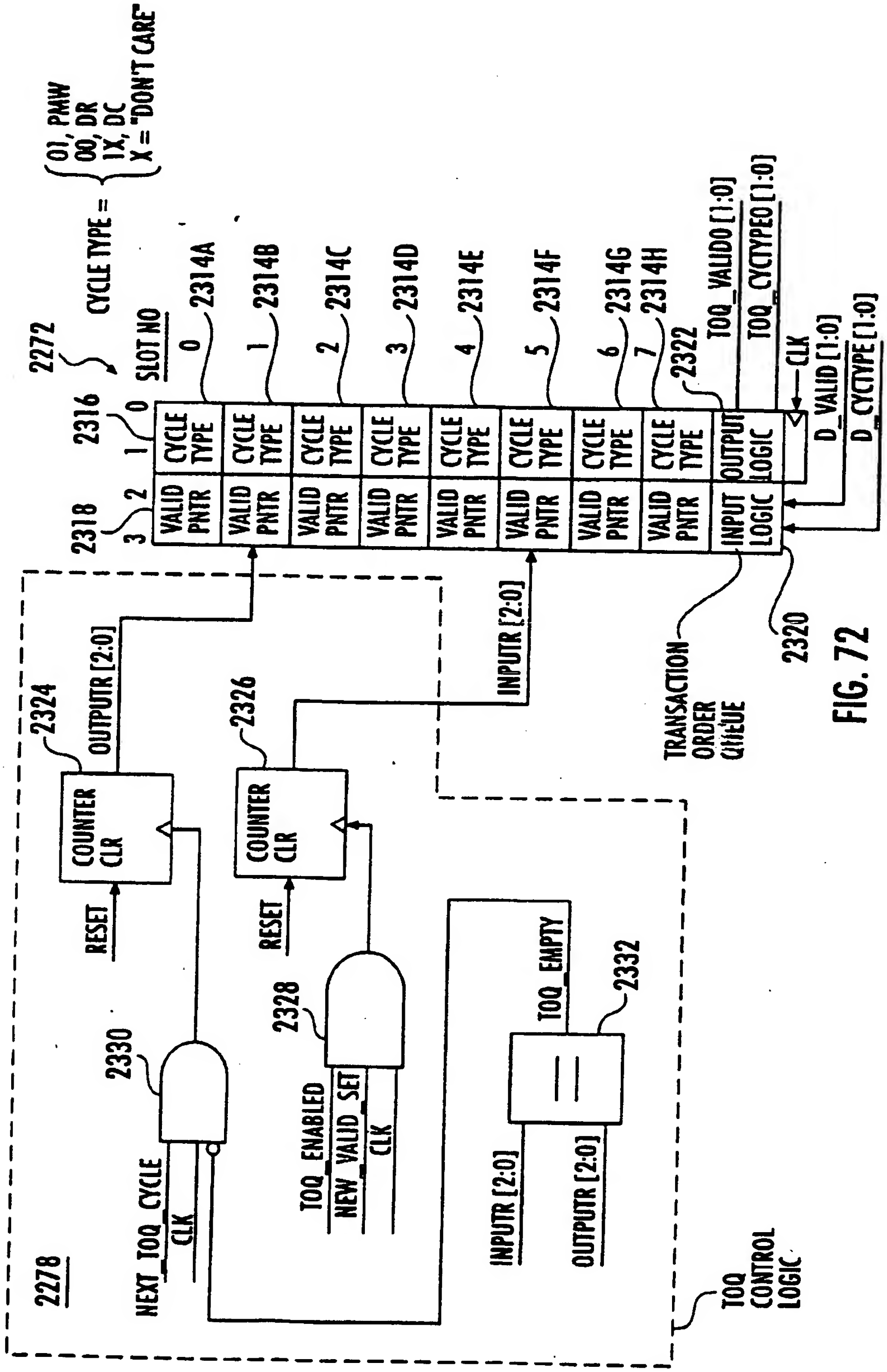
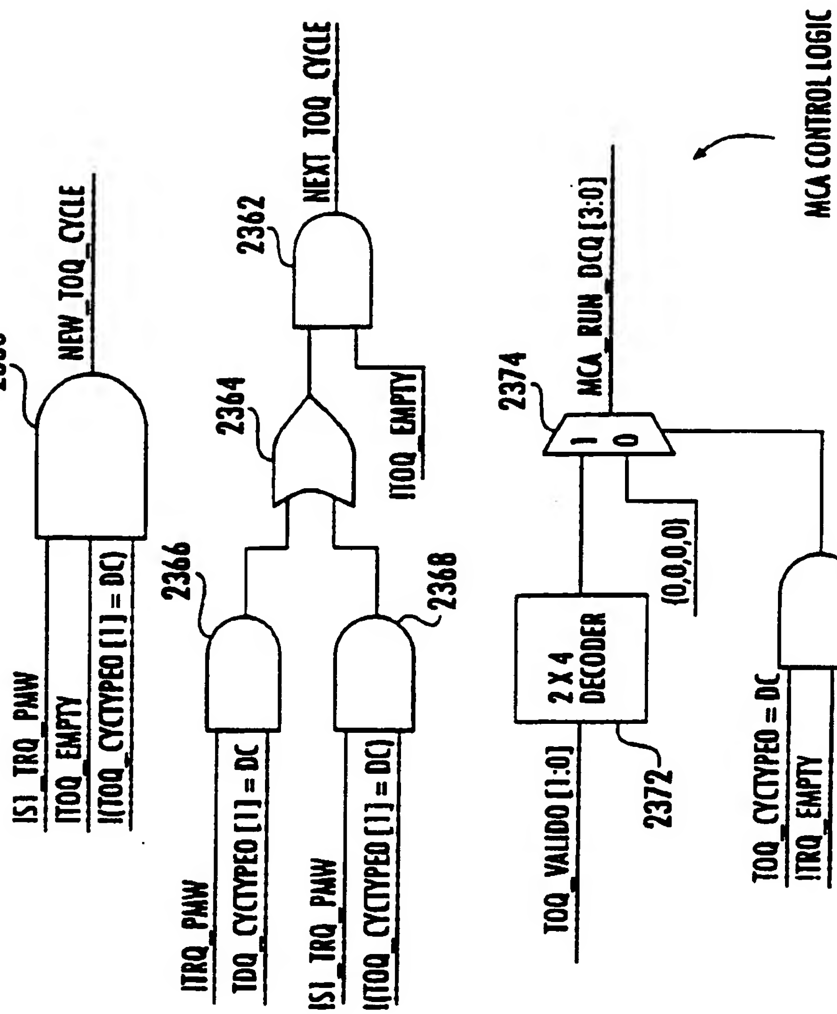

2272
CYCLE TYPE =
01, PMW
00, DR
1X, DC
X = "DON'T CARE"
SLOT NO
0
1
2
3
4
5
6
7
2314A
2314B
2314C
2314D
2314E
2314F
2314G
2314H
2316
2318
2322
2320
VALID PNTR
CYCLE TYPE
INPUT LOGIC
OUTPUT LOGIC
TOQ_VALID0 [1:0]
TOQ_CYCTYPE0 [1:0]
CLK
D_VALID [1:0]
D_CYCTYPE [1:0]
TRANSACTION ORDER QUEUE
OUTPUTR [2:0]
INPUTR [2:0]
2324
2326
COUNTER CLR
RESET
2328
2330
2332
TOQ_EMPTY
NEXT_TOQ_CYCLE
CLK
TOQ_ENABLED
NEW_VALID_SET
CLK
INPUTR [2:0]
OUTPUTR [2:0]
2278
TOQ CONTROL LOGIC

FIG. 72



MCA CONTROL LOGIC
2360
NEW_TOQ_CYCLE
IS1_TRQ_PMW
!TOQ_EMPTY
!(TOQ_CYCTYPEO [1] = DC)
2362
NEXT_TOQ_CYCLE
2364
!TOQ_EMPTY
2366
!TRQ_PMW
TOQ_CYCTYPEO [1] = DC
2368
IS1_TRQ_PMW
!(TOQ_CYCTYPEO [1] = DC)
2374
MCA_RUN_DCQ [3:0]
1
0
{0,0,0,0}
2 X 4 DECODER
2372
TOQ_VALIDO [1:0]
2370
TOQ_CYCTYPEO == DC
!TRQ_EMPTY
!TRQ_PMW

FIG. 73B

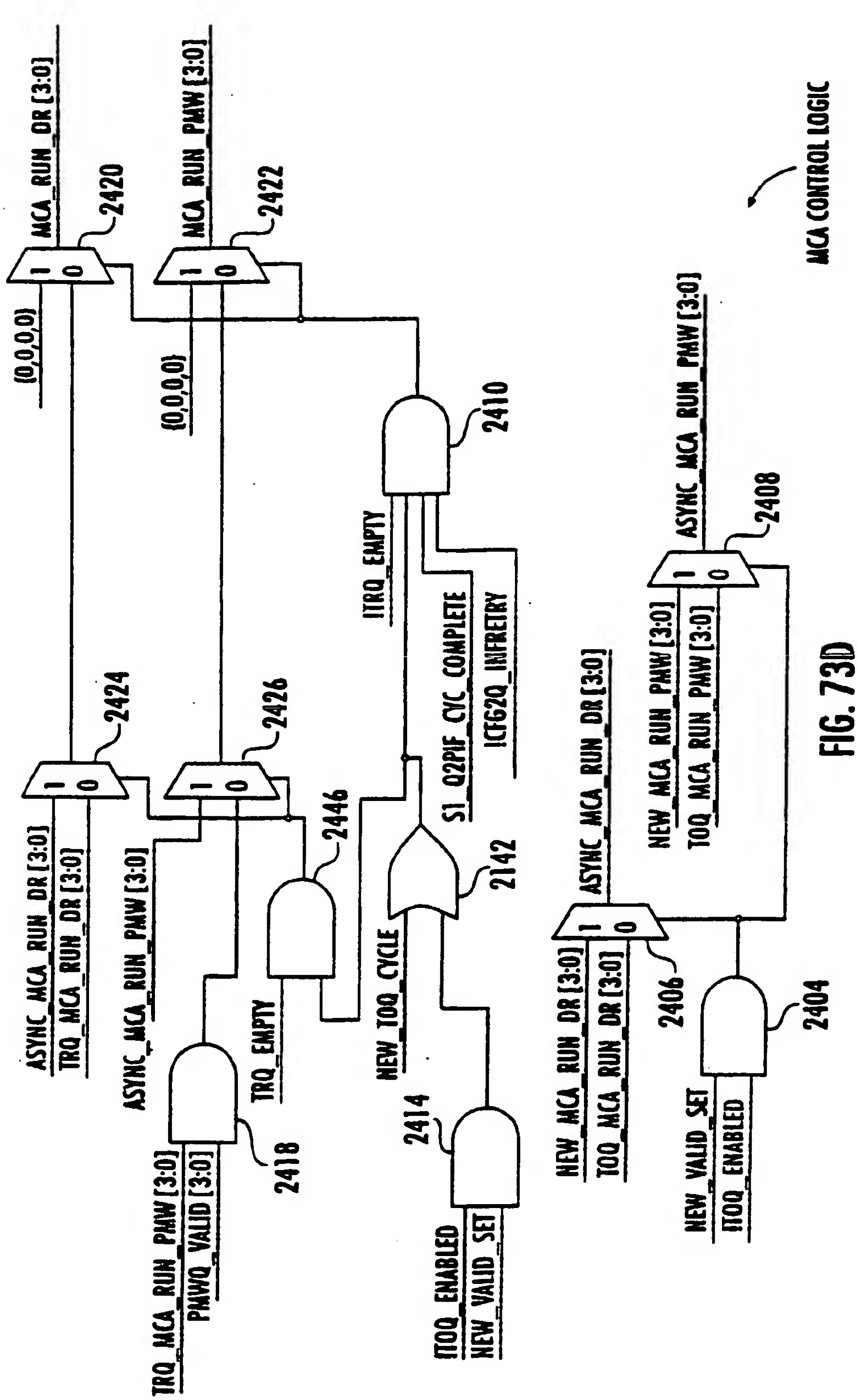

MCA_RUN_DR [3:0]
2420
MCA_RUN_PMW [3:0]
2422
{0,0,0,0}
{0,0,0,0}
1
0
2410
ITRQ_EMPTY
S1_Q2PIF_CYC_COMPLETE
ICFG2Q_INFRETRY
2424
2426
ASYNC_MCA_RUN_DR [3:0]
TRQ_MCA_RUN_DR [3:0]
ASYNC_MCA_RUN_PMW [3:0]
2446
TRQ_EMPTY
2418
TRQ_MCA_RUN_PMW [3:0]
PMWQ_VALID [3:0]
2142
NEW_TOQ_CYCLE
2414
ITOQ_ENABLED
NEW_VALID_SET
ASYNC_MCA_RUN_DR [3:0]
NEW_MCA_RUN_DR [3:0]
TOQ_MCA_RUN_DR [3:0]
2406
2404
NEW_VALID_SET
ITOQ_ENABLED
ASYNC_MCA_RUN_PMW [3:0]
2408
NEW_MCA_RUN_PMW [3:0]
TOQ_MCA_RUN_PMW [3:0]
MCA CONTROL LOGIC

FIG. 73D

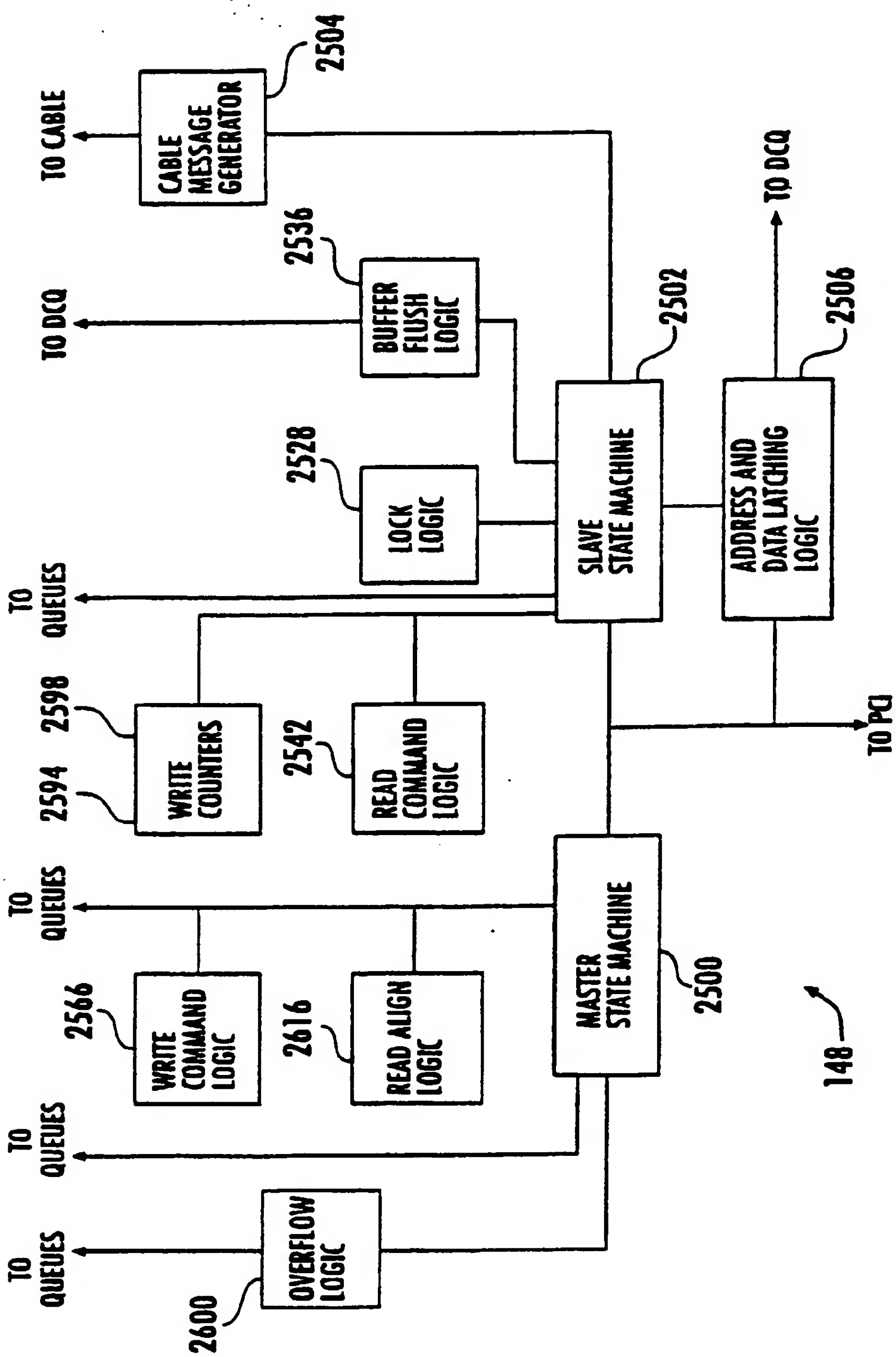

TO CABLE
CABLE MESSAGE GENERATOR
2504
TO DCQ
BUFFER FLUSH LOGIC
2536
TO DCQ
2502
2506
LOCK LOGIC
2528
SLAVE STATE MACHINE
ADDRESS AND DATA LATCHING LOGIC
TO QUEUES
2594
2598
WRITE COUNTERS
2542
READ COMMAND LOGIC
TO PCI
TO QUEUES
MASTER STATE MACHINE
2500
2566
WRITE COMMAND LOGIC
2616
READ ALIGN LOGIC
148
TO QUEUES
TO QUEUES
OVERFLOW LOGIC
2600

FIG. 75

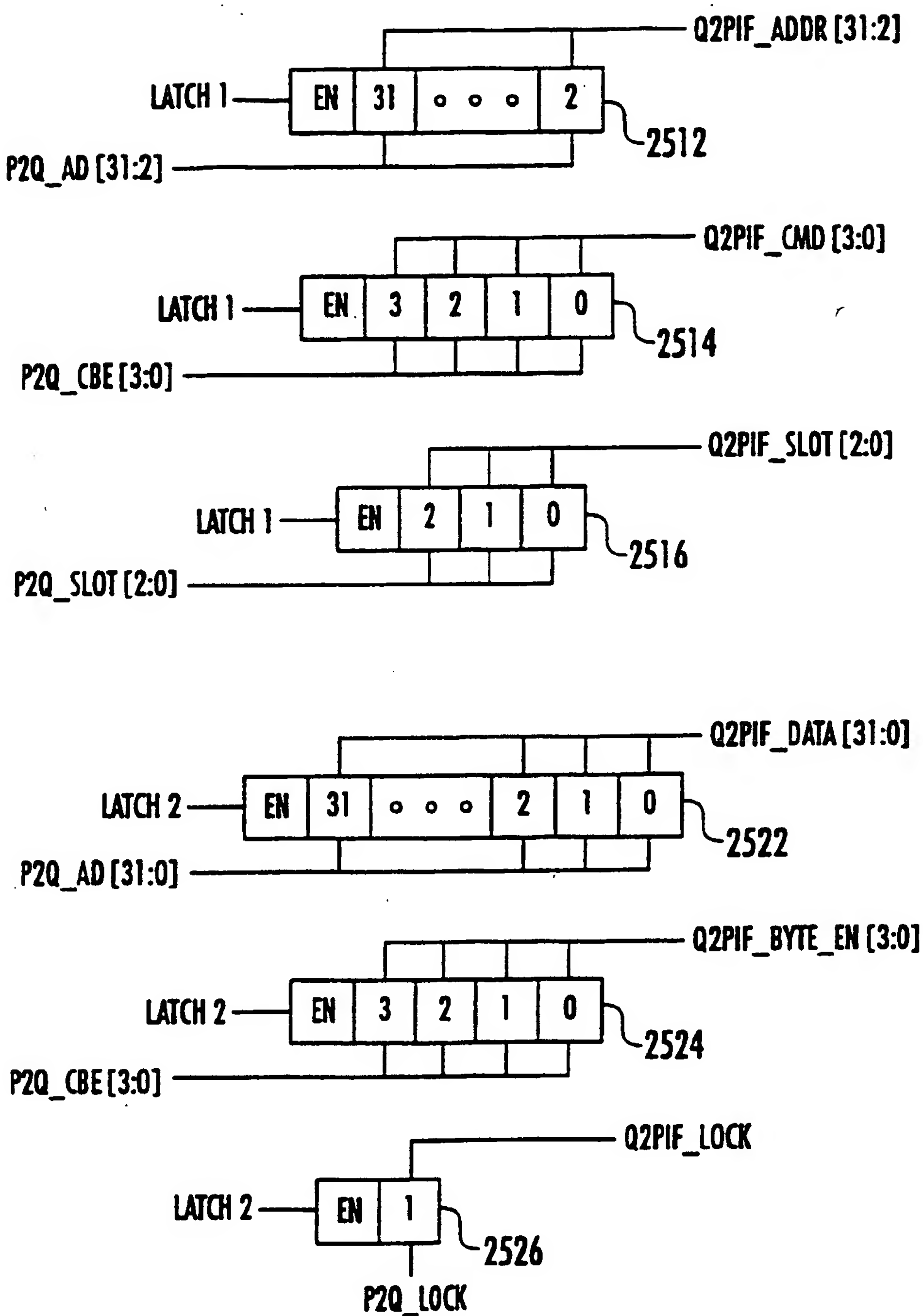

Q2PIF_ADDR [31:2]
LATCH 1
EN
31
2
2512
P2Q_AD [31:2]
Q2PIF_CMD [3:0]
LATCH 1
EN
3
2
1
0
2514
P2Q_CBE [3:0]
Q2PIF_SLOT [2:0]
LATCH 1
EN
2
1
0
2516
P2Q_SLOT [2:0]
Q2PIF_DATA [31:0]
LATCH 2
EN
31
2
1
0
2522
P2Q_AD [31:0]
Q2PIF_BYTE_EN [3:0]
LATCH 2
EN
3
2
1
0
2524
P2Q_CBE [3:0]
Q2PIF_LOCK
LATCH 2
EN
1
2526
P2Q_LOCK

FIG. 76B

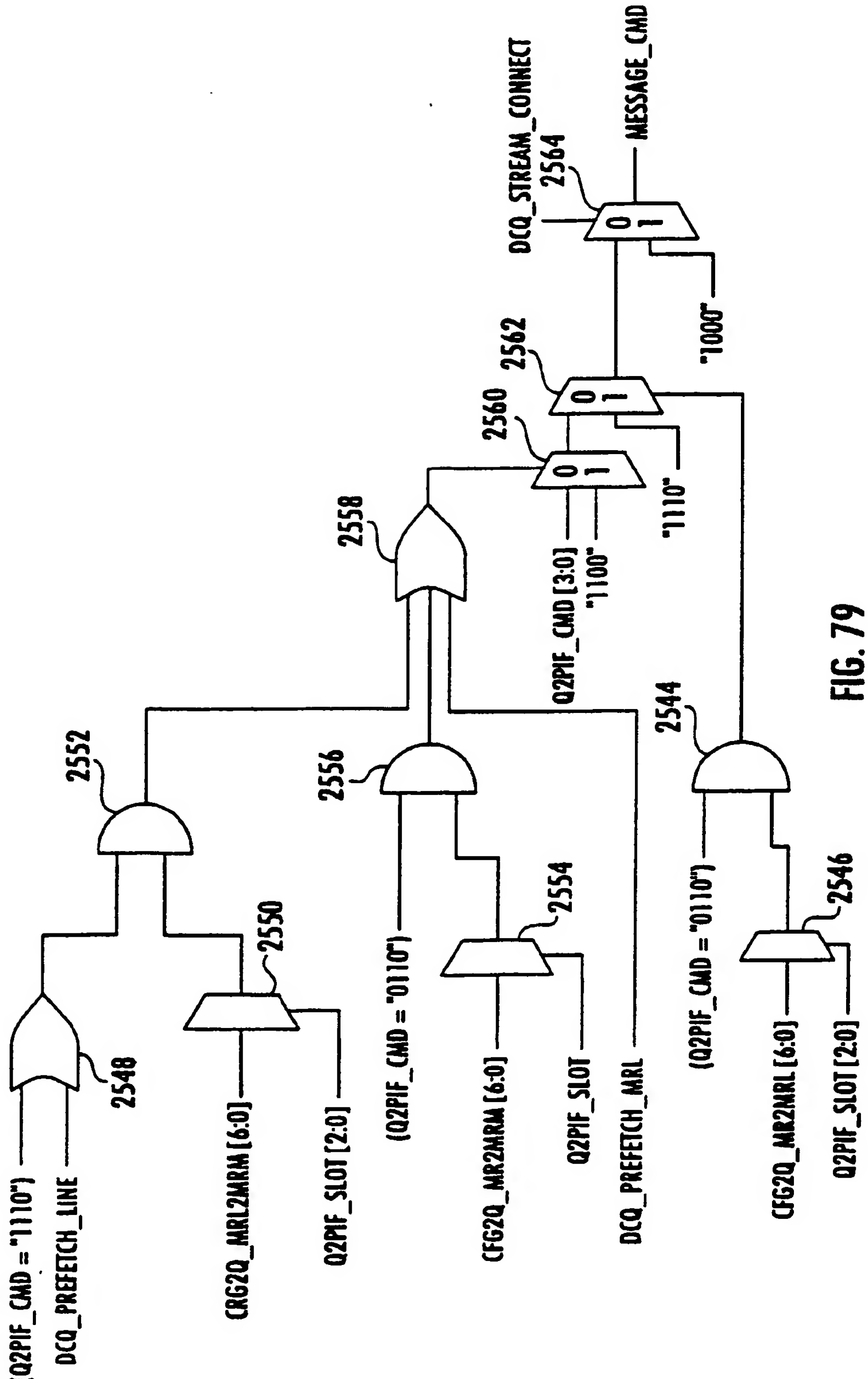

(Q2PIF_CMD = "1110")
DCQ_PREFETCH_LINE
2548
CRG2Q_MRL2MRM [6:0]
Q2PIF_SLOT [2:0]
2550
2552
(Q2PIF_CMD = "0110")
CFG2Q_MR2MRM [6:0]
Q2PIF_SLOT
2554
2556
DCQ_PREFETCH_MRL
2558
Q2PIF_CMD [3:0]
"1100"
2560
0
1
"1110"
2562
0
1
"1000"
2564
0
1
DCQ_STREAM_CONNECT
MESSAGE_CMD
(Q2PIF_CMD = "0110")
CFG2Q_MR2MRL [6:0]
Q2PIF_SLOT [2:0]
2546
2544

FIG. 79

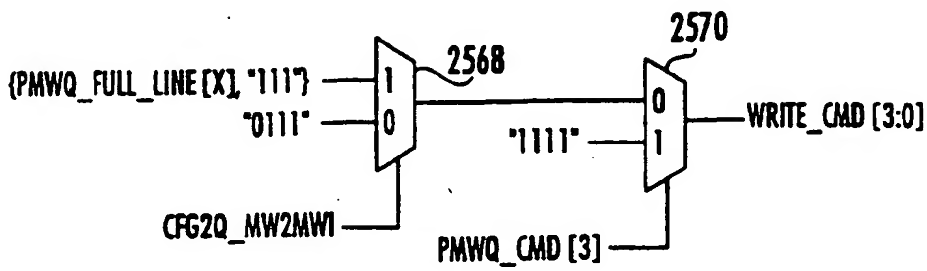


FIG. 81A

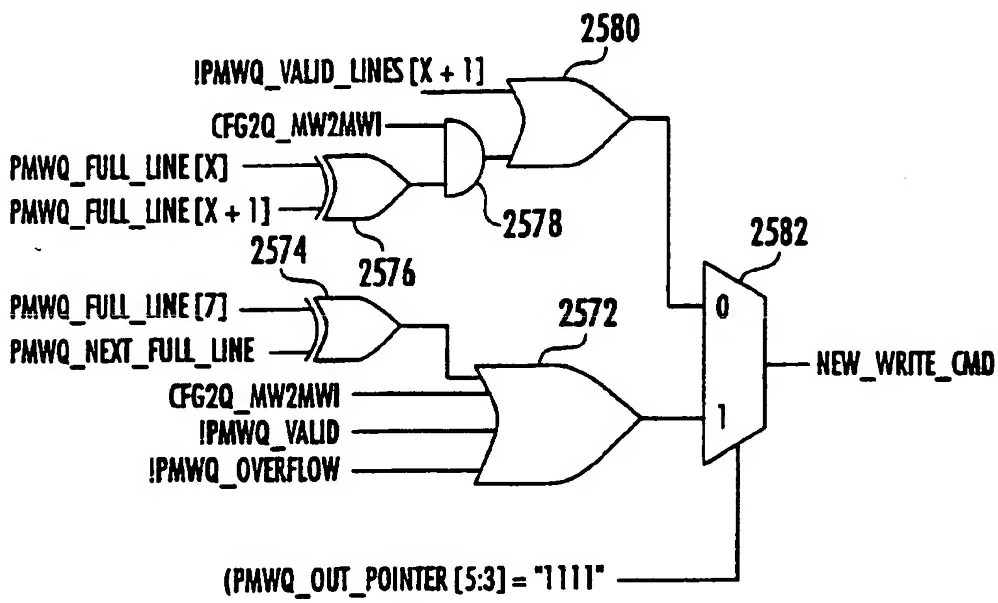


FIG. 81B

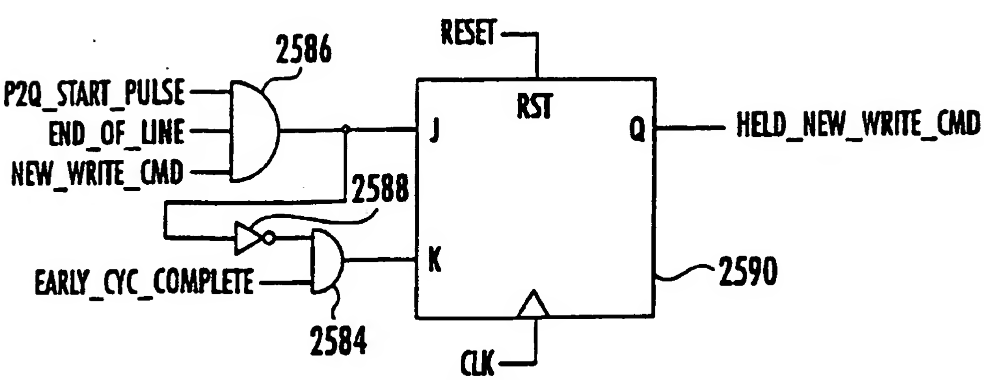


FIG. 81C

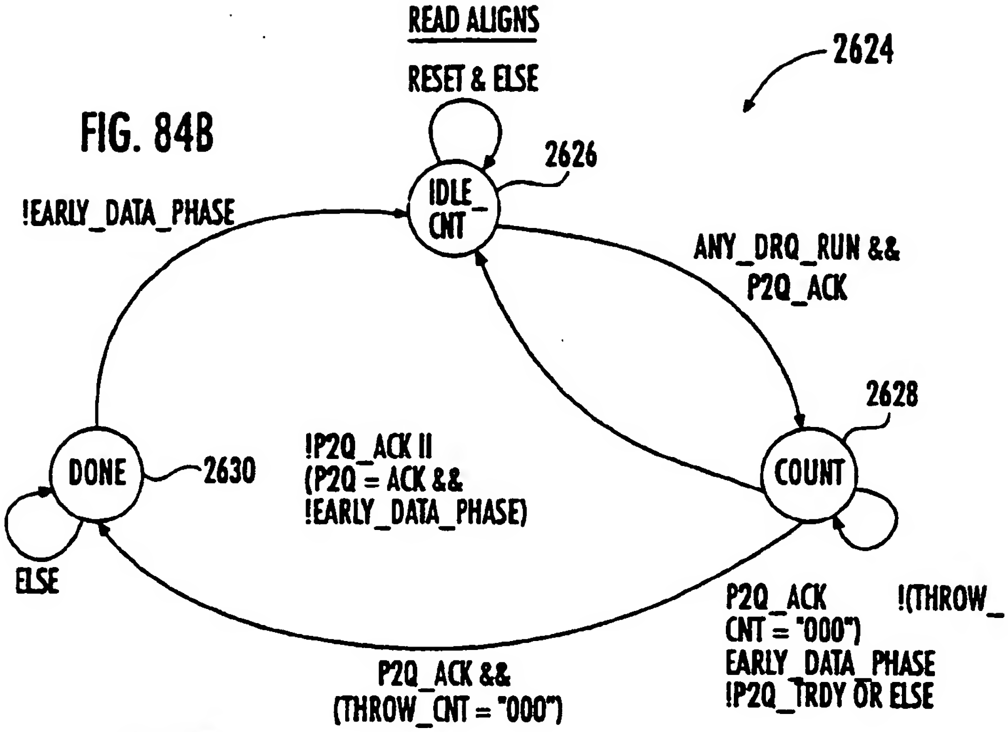


FIG. 84B

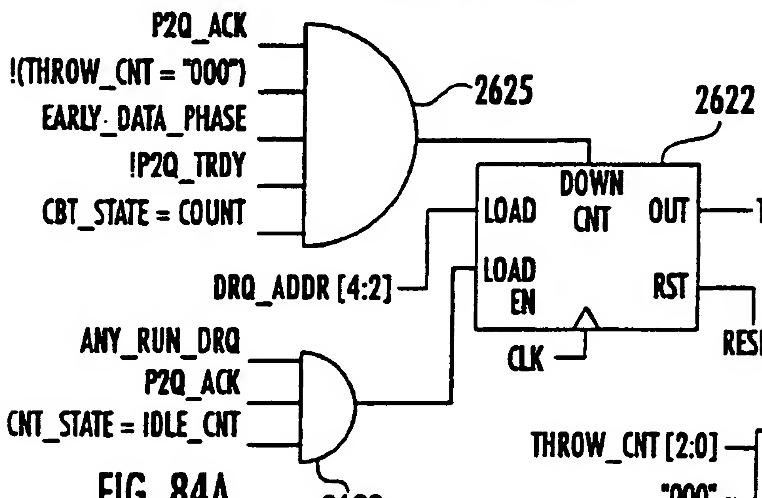


FIG. 84A

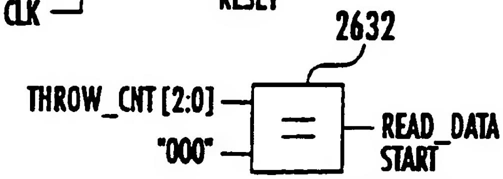


FIG. 84C

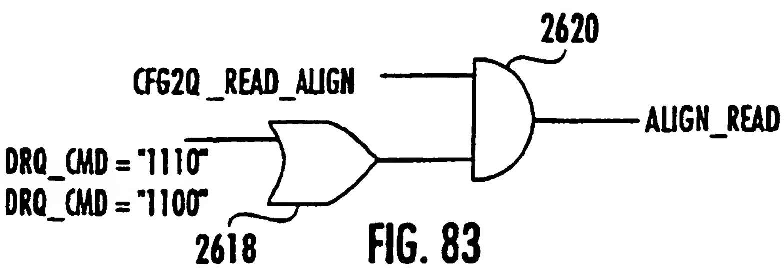


FIG. 83

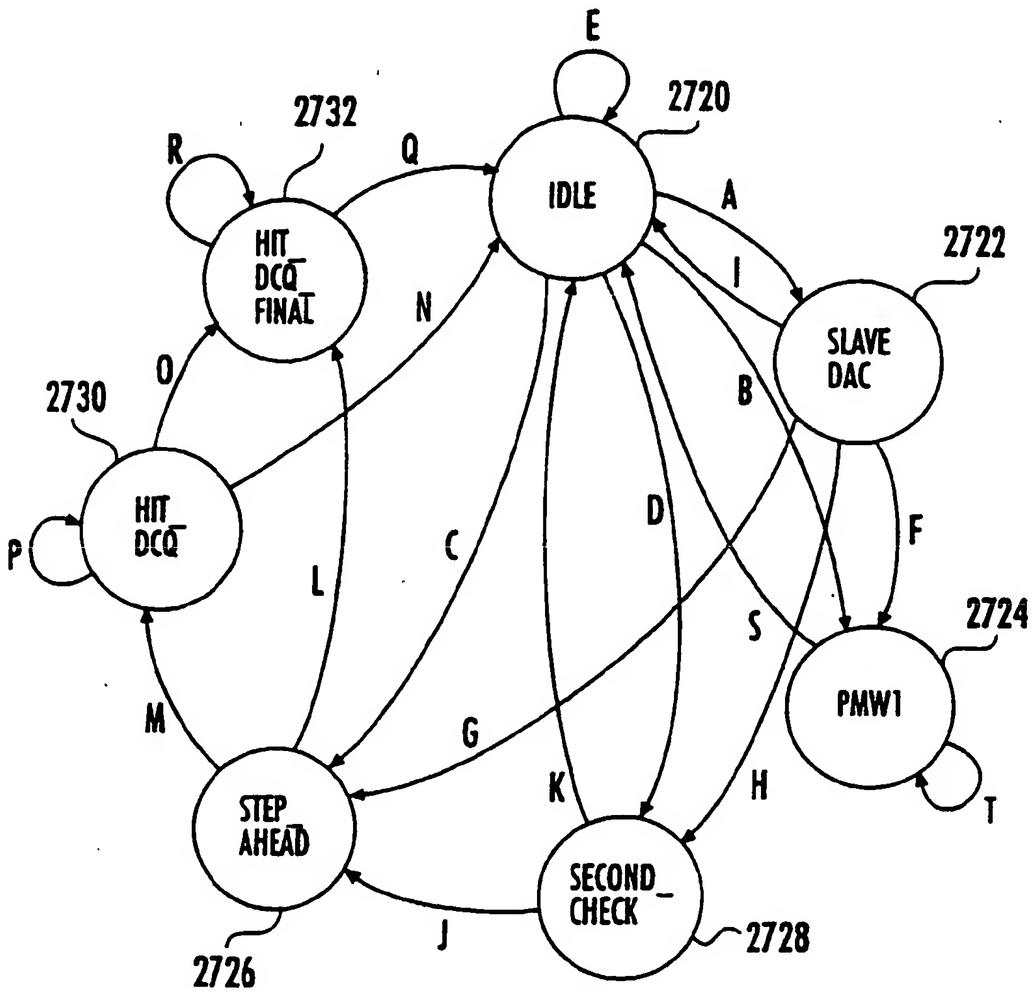
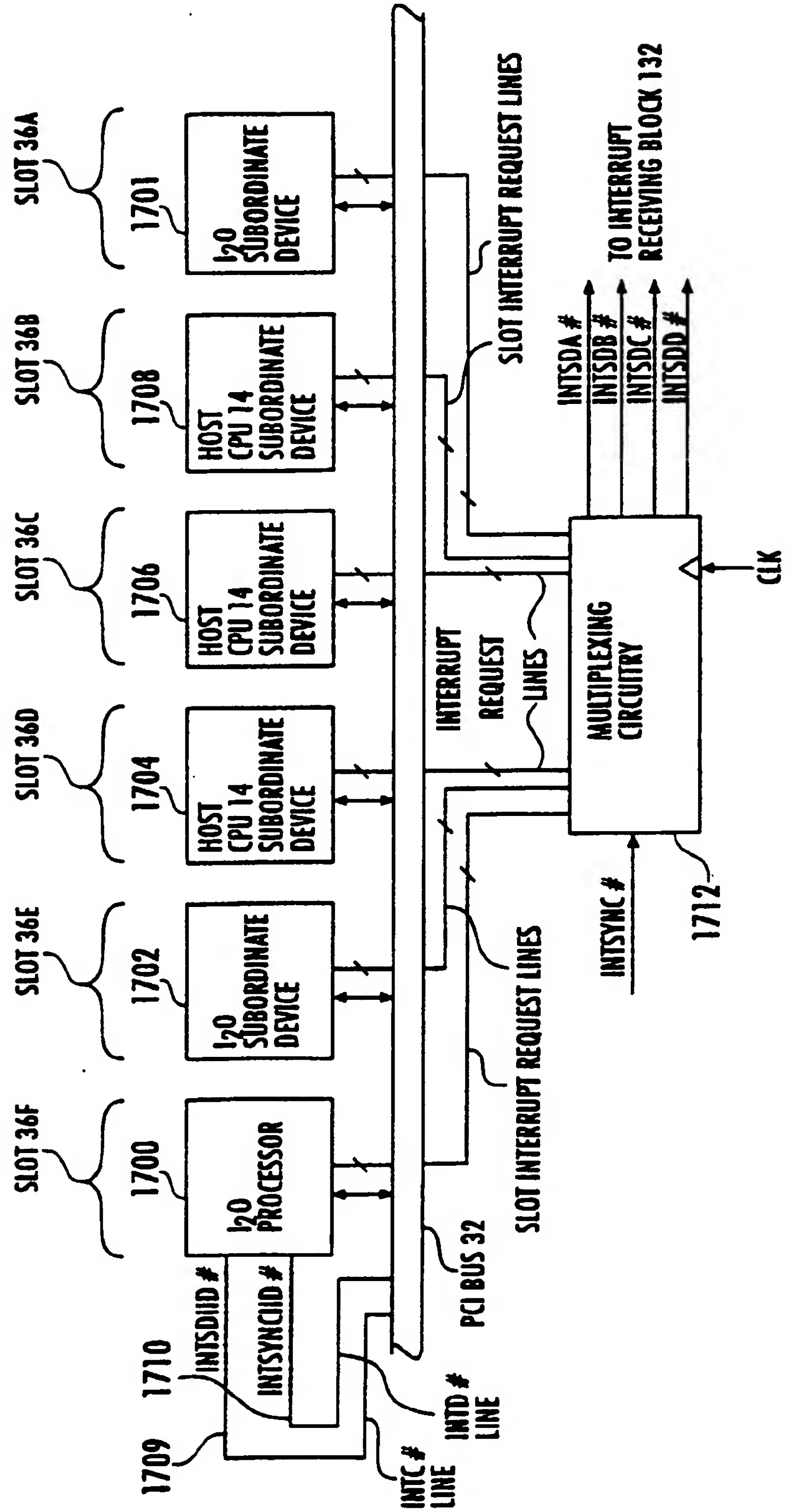

E
2720
IDLE
2732
R
Q
HIT_
DCQ_
FINAL
A
I
2722
SLAVE
DAC
N
O
2730
B
HIT_
DCQ
P
L
C
D
F
S
2724
PMW1
M
G
K
H
T
STEP_
AHEAD
SECOND_
CHECK
J
2728
2726

FIG. 86



SLOT 36A
SLOT 36B
SLOT 36C
SLOT 36D
SLOT 36E
SLOT 36F
1701
I2O SUBORDINATE DEVICE
1708
HOST CPU 14 SUBORDINATE DEVICE
1706
HOST CPU 14 SUBORDINATE DEVICE
1704
HOST CPU 14 SUBORDINATE DEVICE
1702
I2O SUBORDINATE DEVICE
1700
I2O PROCESSOR
SLOT INTERRUPT REQUEST LINES
INTERRUPT REQUEST LINES
SLOT INTERRUPT REQUEST LINES
TO INTERRUPT RECEIVING BLOCK 132
INTSDA #
INTSDB #
INTSDC #
INTSDD #
MULTIPLEXING CIRCUITRY
CLK
INTSYNC #
1712
PCI BUS 32
INTSDIID #
INTSYNCIID #
1710
1709
INTD # LINE
INTC # LINE

FIG. 88

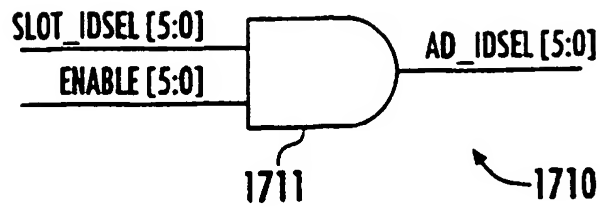


FIG. 90

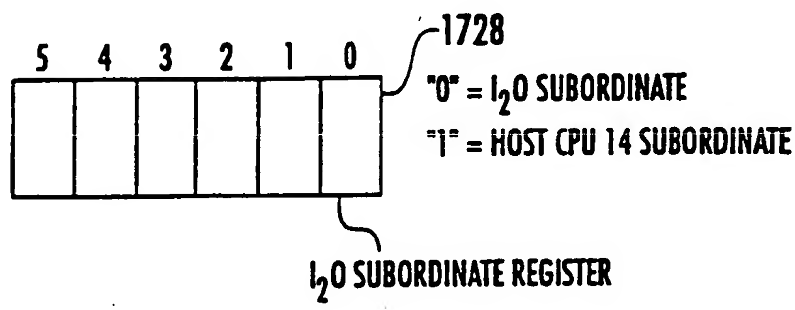


FIG. 91

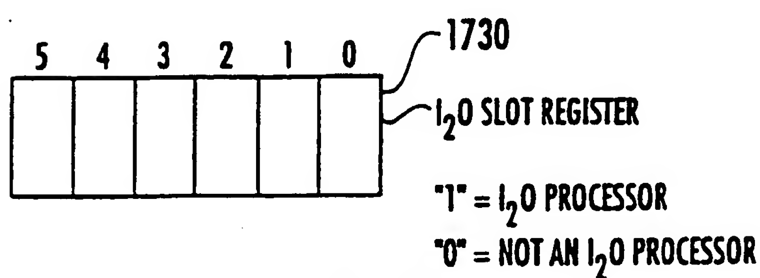


FIG. 92

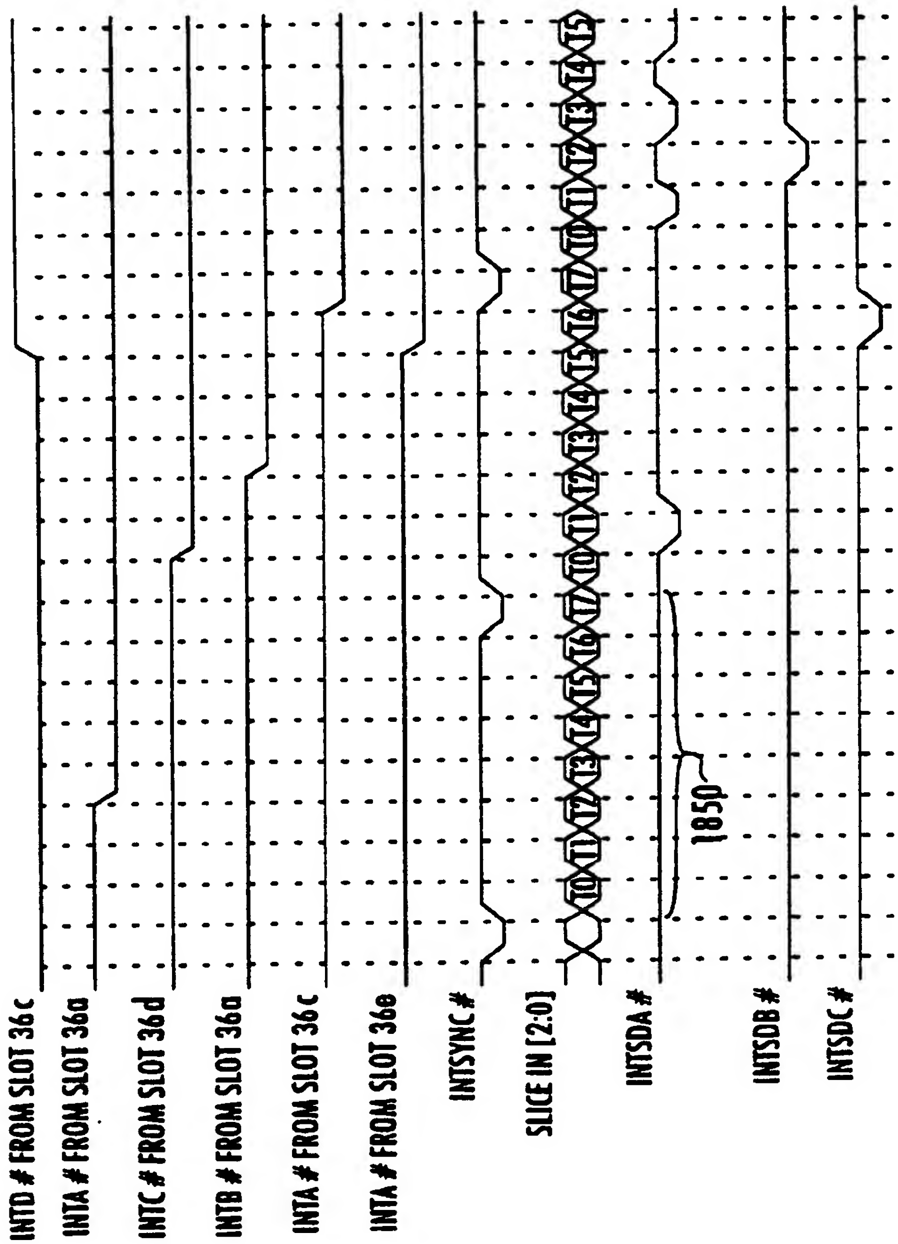


FIG. 95A

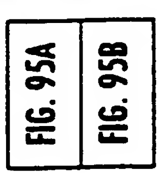


FIG. 95

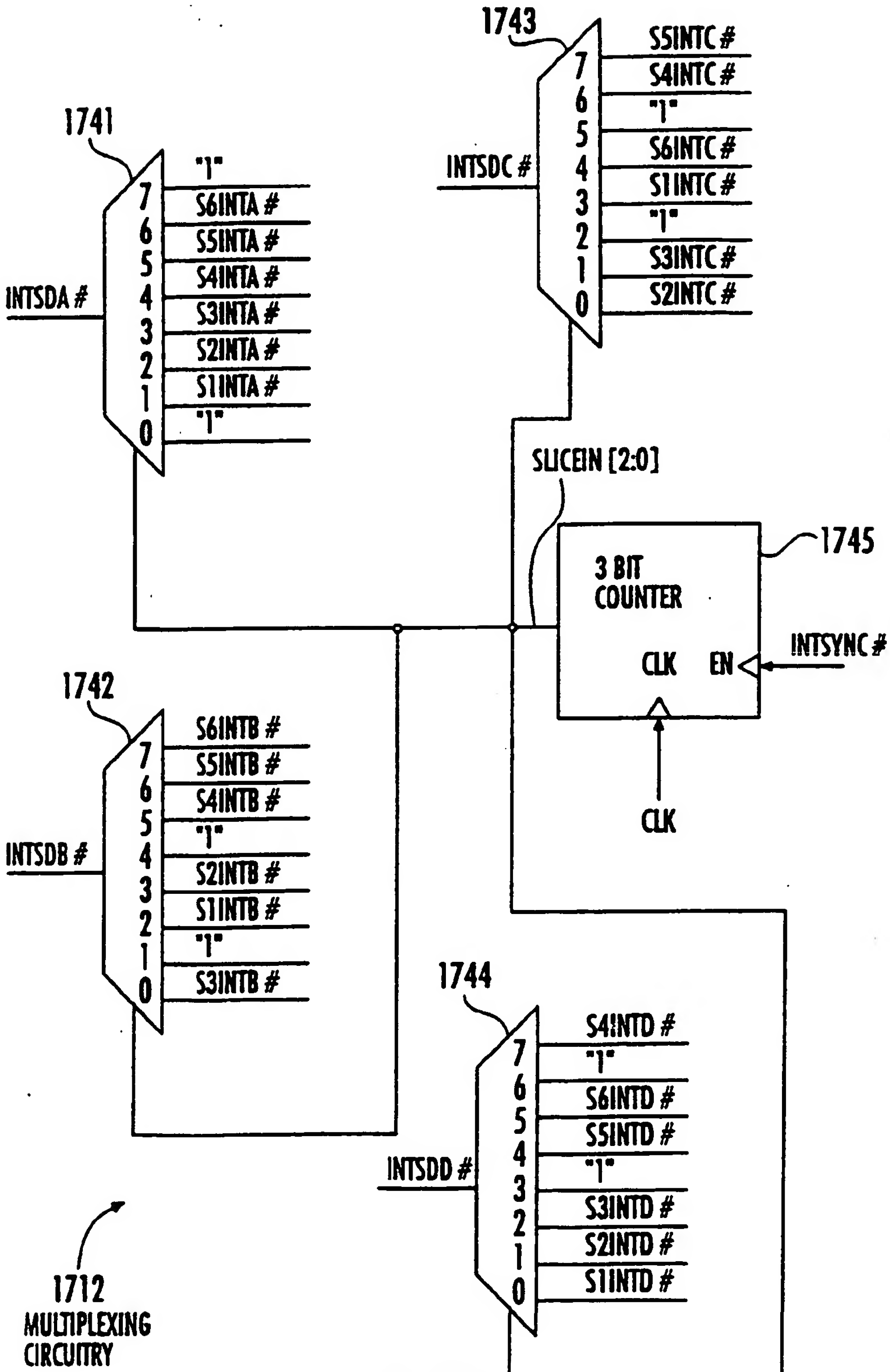

1743
INTSDC #
7
6
5
4
3
2
1
0
S5INTC #
S4INTC #
"1"
S6INTC #
S1INTC #
"1"
S3INTC #
S2INTC #
1741
INTSDA #
"1"
S6INTA #
S5INTA #
S4INTA #
S3INTA #
S2INTA #
S1INTA #
"1"
SLICEIN [2:0]
3 BIT COUNTER
1745
CLK
EN
INTSYNC #
CLK
1742
INTSDB #
S6INTB #
S5INTB #
S4INTB #
"1"
S2INTB #
S1INTB #
"1"
S3INTB #
1744
INTSDD #
S4INTD #
"1"
S6INTD #
S5INTD #
"1"
S3INTD #
S2INTD #
S1INTD #
1712
MULTIPLEXING CIRCUITRY

FIG. 96

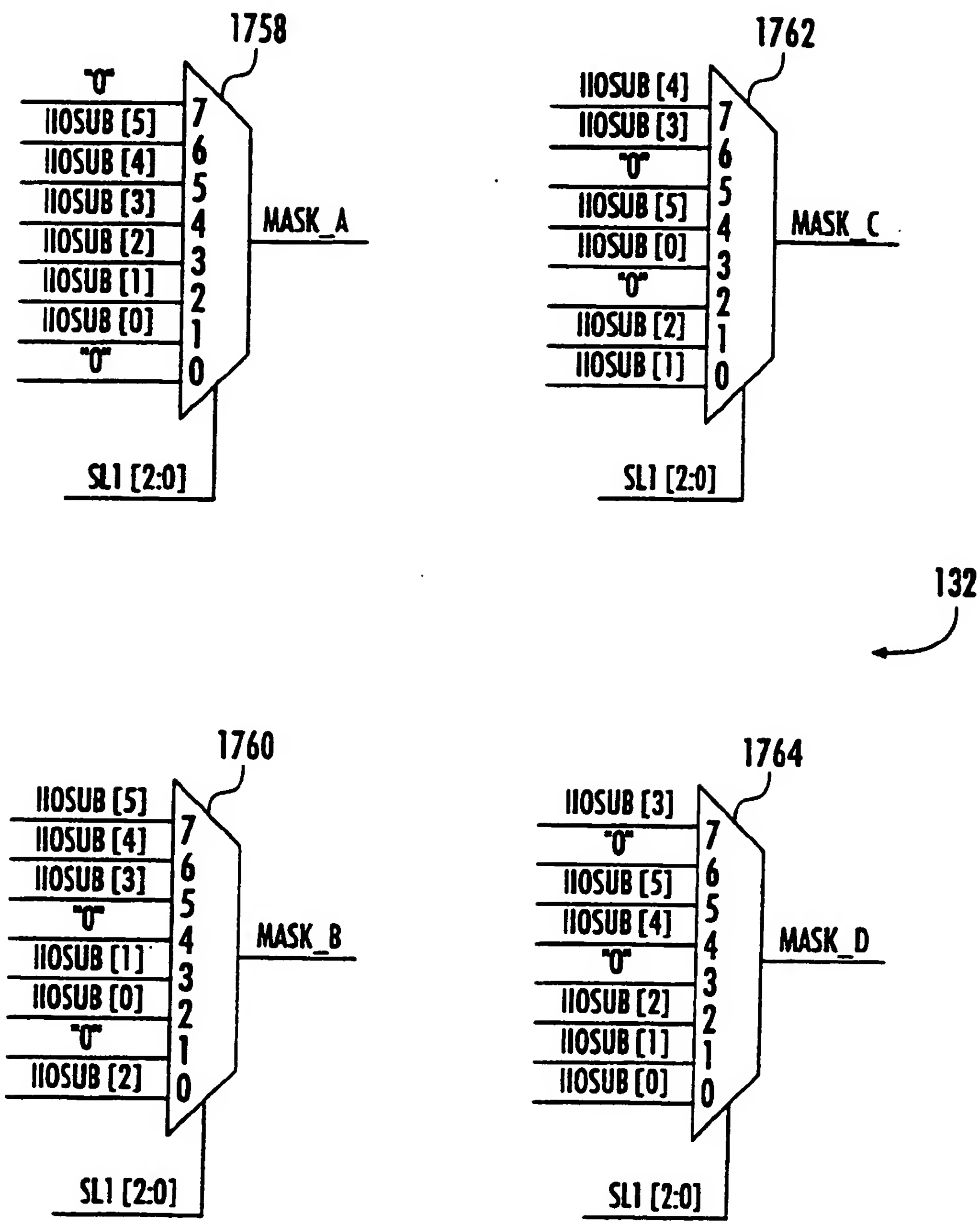

1758
"0"
IIOSUB [5]
IIOSUB [4]
IIOSUB [3]
IIOSUB [2]
IIOSUB [1]
IIOSUB [0]
"0"
7
6
5
4
3
2
1
0
MASK_A
SL1 [2:0]
1762
IIOSUB [4]
IIOSUB [3]
"0"
IIOSUB [5]
IIOSUB [0]
"0"
IIOSUB [2]
IIOSUB [1]
7
6
5
4
3
2
1
0
MASK_C
SL1 [2:0]
132
1760
IIOSUB [5]
IIOSUB [4]
IIOSUB [3]
"0"
IIOSUB [1]
IIOSUB [0]
"0"
IIOSUB [2]
7
6
5
4
3
2
1
0
MASK_B
SL1 [2:0]
1764
IIOSUB [3]
"0"
IIOSUB [5]
IIOSUB [4]
"0"
IIOSUB [2]
IIOSUB [1]
IIOSUB [0]
7
6
5
4
3
2
1
0
MASK_D
SL1 [2:0]

FIG. 97B

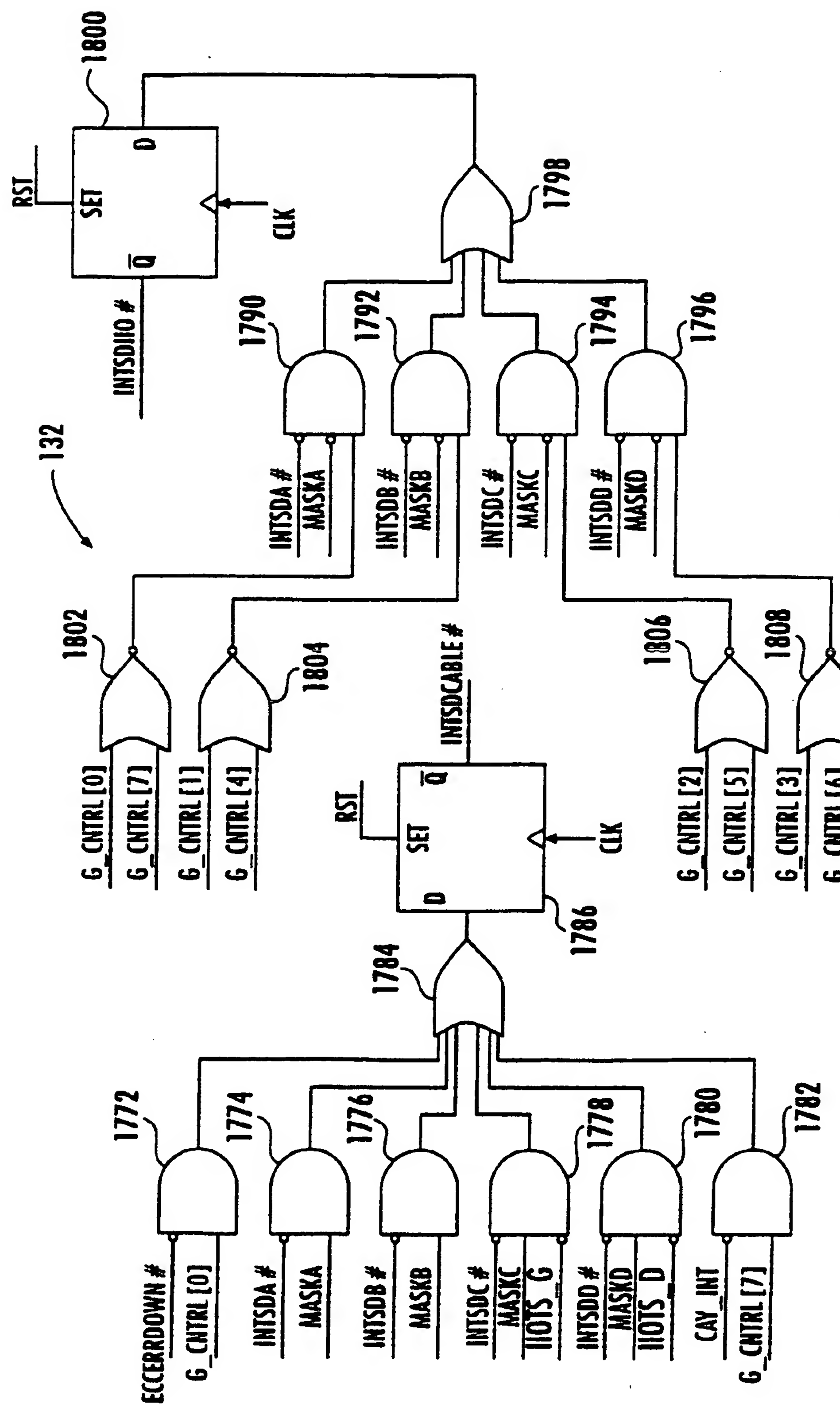

132
1800
RST
SET
D
Q̄
CLK
INTSDIIO #
1798
1790
1792
1794
1796
INTSDA #
MASKA
INTSDB #
MASKB
INTSDC #
MASKC
INTSDD #
MASKD
1802
1804
1806
1808
G_CNTRL [0]
G_CNTRL [7]
G_CNTRL [1]
G_CNTRL [4]
G_CNTRL [2]
G_CNTRL [5]
G_CNTRL [3]
G_CNTRL [6]
INTSDCABLE #
1786
1784
1772
1774
1776
1778
1780
1782
ECCERRDOWN #
IIOTS_G
IIOTS_D
CAY_INT

FIG. 97D

TIME SLICE	INTSDA #	INTSDD #	INTSDC #	INTSDB #
T0		SLOT 1. INTD #	SLOT 2. INTC #	SLOT 3. INTB #
T1	SLOT 1. INTA #	SLOT 2. INTD #	SLOT 3. INTC #	
T2	SLOT 2. INTA #	SLOT 3. INTD #		SLOT 1. INTB #
T3	SLOT 3. INTA #		SLOT 1. INTC #	SLOT 2. INTB #
T4	SLOT 4. INTA #	SLOT 5. INTD #	SLOT 6. INTC #	
T5	SLOT 5. INTA #	SLOT 6. INTD #		SLOT 4. INTB #
T6	SLOT 6. INTA #		SLOT 4. INTC #	SLOT 5. INTB #
T7		SLOT 4. INTD #	SLOT 5. INTC #	SLOT 6. INTB #

FIG. 99

SLOT 1 = SLOT 36A

SLOT 2 = SLOT 36B

SLOT 3 = SLOT 36C

SLOT 4 = SLOT 36D

SLOT 5 = SLOT 36E

SLOT 6 = SLOT 36F

INTERRUPT LINES ON PCI BUS 24	EXPANSION BUS 30 INTERRUPT SOURCES							
INTA #	BR_INTR #	SLOT 1. INTD #	SLOT 2. INTC #	SLOT 3. INTB #	SLOT 4. INTA #	SLOT 5. INTD #	SLOT 6. INTC #	"1"
INTB #	SLOT 1. INTA #	SLOT 2. INTD #	SLOT 3. INTC #	"1"	SLOT 5. INTA #	SLOT 6. INTD #	"1"	SLOT 4. INTB #
INTC #	SLOT 2. INTA #	SLOT 3. INTD #	"1"	SLOT 1. INTB #	SLOT 6. INTA #	"1"	SLOT 4. INTC #	SLOT 5. INTB #
INTD #	SLOT 3. INTA #	"1"	SLOT 1. INTC #	SLOT 2. INTB #	SI_ INTR #	SLOT 4. INTD #	SLOT 5. INTC #	SLOT 6. INTB #

FIG. 100

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 811 931 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
29.12.1997 Bulletin 1997/52

(51) Int Cl.6: G06F 13/40

(43) Date of publication A2:
10.12.1997 Bulletin 1997/50

(21) Application number: 97303797.1

(22) Date of filing: 04.06.1997

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

(30) Priority: 05.06.1996 US 658602

(71) Applicant: Compaq Computer Corporation
Houston Texas 77070 (US)

(72) Inventors:
- Culley, Paul R.
Cypress,Texas 77429 (US)
- Goodrum, Alan L.
Tomball, Texas 77375 (US)
- Chow, Raymond Y. L.
Cypress, Texas 77429 (US)
- Basile, Barry S.
Houston, Texas 77084 (US)

(74) Representative: Brunner, Michael John
GILL JENNINGS & EVERY
Broadgate House
7 Eldon Street
London EC2M 7LH (GB)

(54) **Expansion card insertion and removal**

(57) A computer system has a bus, a connector for a circuit card, and a clamp configured to selectively prevent removal of the circuit card from the connector when the clamp is engaged. The computer system has circuitry connected to monitor the engagement status of the clamp and to regulate delivery of power to the connector based on the engagement state of the clamp.

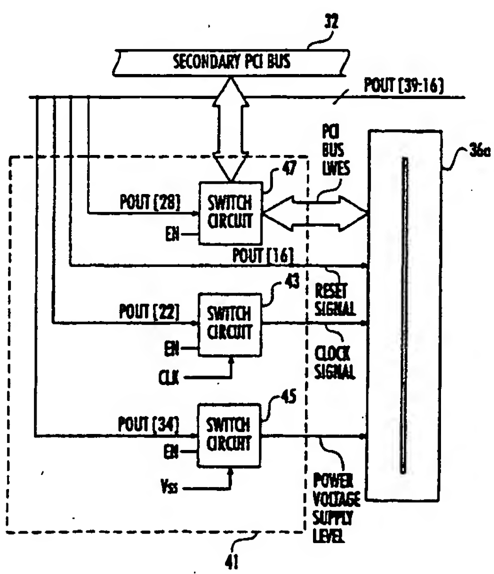


FIG. 28

Printed by Jouve, 75001 PARIS (FR)

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 97 30 3797

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X Y	EP 0 254 456 A (AT & T) * page 1, column 1, line 13 - page 3, column 3, line 10; figures 1,2 *	1,5, 7-14,18, 20,21, 23-25 2-4,6, 15-17, 19,22	G06F13/40
Y	WO 93 15459 A (MICRO INDUSTRIES) * page 12, paragraph 2 - page 16, paragraph 3; claim 4; figures 4,5,8 *	2-4,6, 15-17, 19,22	
A	"Circuits to Allow Cartridge Hot-Plugging" IBM TECHNICAL DISCLOSURE BULLETIN., vol. 29, no. 7, December 1986, NEW YORK US, pages 2877-2878, XP002040915 * the whole document *	2,15	
			TECHNICAL FIELDS SEARCHED (Int.Cl.6)
			G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	25 September 1997	Gill, S

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document